hostapd Reference Manual

0.5.x

Generated by Doxygen 1.4.2

Sun Dec 31 13:43:25 2006

Contents

Chapter 1

Developers' documentation for hostapd

hostapd includes IEEE 802.11 access point management (authentication / association), IEEE 802.1X/WPA/WPA2 Authenticator, EAP server, and RADIUS authentication server functionality. It can be build with various configuration option, e.g., a standalone AP management solution or a RADIUS authentication server with support for number of EAP methods.

The goal of this documentation and comments in the source code is to give enough information for other developers to understand how hostapd has been implemented, how it can be modified, how new drivers can be supported, and how hostapd can be ported to other operating systems. If any information is missing, feel free to contact Jouni Malinen <`jkmaline@cc.hut.fi`> for more information. Contributions as patch files are also very welcome at the same address. Please note that hostapd is licensed under dual license, GPLv2 or BSD at user's choice. All contributions to hostapd are expected to use compatible licensing terms.

The source code and read-only access to hostapd CVS repository is available from the project home page at `http://hostap.epitest.fi/hostapd/`. This developers' documentation is also available as a PDF file from `http://hostap.epitest.fi/hostapd/hostapd-devel.pdf` .

The design goal for hostapd was to use hardware, driver, and OS independent, portable C code for all WPA functionality. The source code is divided into separate C files as shown on the code structure page. All hardware/driver specific functionality is in separate files that implement a well-defined driver API. Information about porting to different target boards and operating systems is available on the porting page.

EAPOL (IEEE 802.1X) state machines are implemented as a separate module that interacts with EAP server implementation. Similarly, RADIUS authentication server is in its own separate module. Both IEEE 802.1X and RADIUS authentication server can use EAP server functionality.

hostapd implements a control interface that can be used by external programs to control the operations of the hostapdt daemon and to get status information and event notifications. There is a small C library that provides helper functions to facilitate the use of the control interface. This library can also be used with C++.

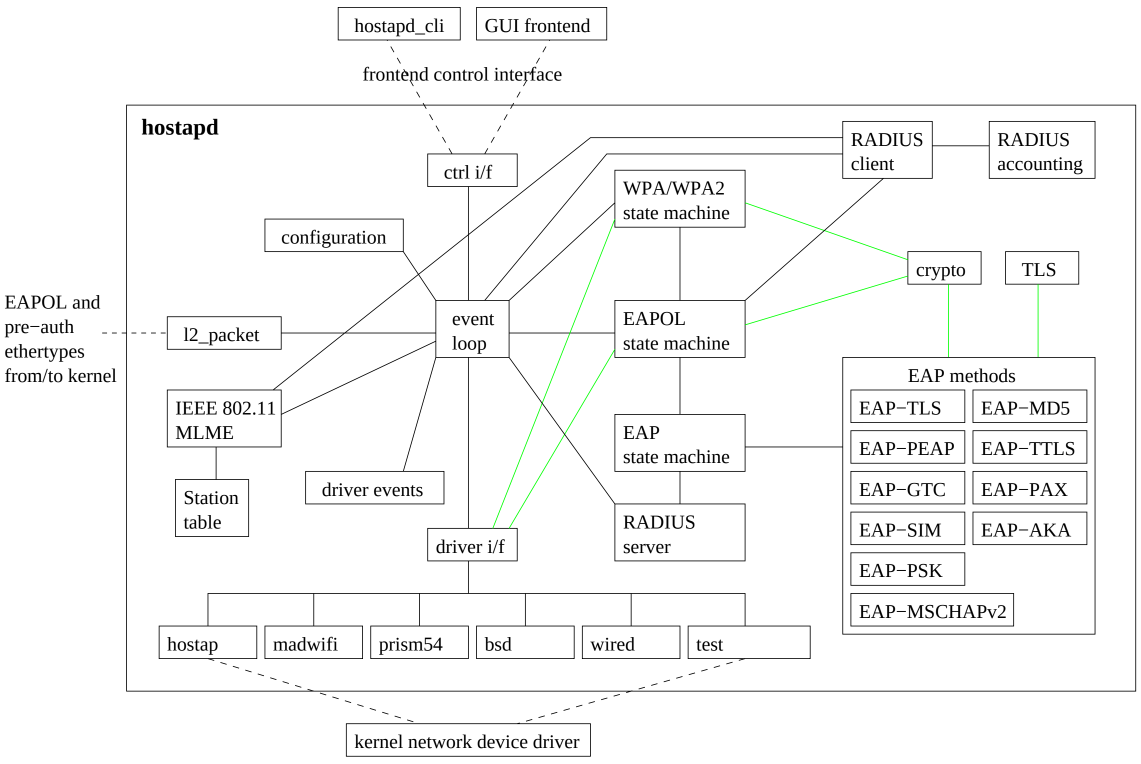


Figure 1.1: hostapd modules

Chapter 2

hostapd Data Structure Index

2.1 hostapd Data Structures

Here are the data structures with brief descriptions:

Chapter 3

hostapd File Index

3.1 hostapd File List

Here is a list of all documented files with brief descriptions:

Chapter 4

hostapd Page Index

4.1 hostapd Related Pages

Here is a list of all related documentation pages:

Chapter 5

hostapd Data Structure Documentation

5.1 eap_method Struct Reference

EAP method interface.

`#include <eap_i.h>`

Collaboration diagram for eap_method:

Data Fields

- int **vendor**
- EapType **method**
- const char ∗ **name**
- void ∗(∗ **init**)(struct eap_sm ∗sm)
- void ∗(∗ **initPickUp**)(struct eap_sm ∗sm)
- void(∗ **reset**)(struct eap_sm ∗sm, void ∗priv)
- u8 ∗(∗ **buildReq**)(struct eap_sm ∗sm, void ∗priv, int id, size_t ∗reqDataLen)
- int(∗ **getTimeout**)(struct eap_sm ∗sm, void ∗priv)
- Boolean(∗ **check**)(struct eap_sm ∗sm, void ∗priv, u8 ∗respData, size_t respDataLen)
- void(∗ **process**)(struct eap_sm ∗sm, void ∗priv, u8 ∗respData, size_t respDataLen)
- Boolean(∗ **isDone**)(struct eap_sm ∗sm, void ∗priv)
- u8 ∗(∗ **getKey**)(struct eap_sm ∗sm, void ∗priv, size_t ∗len)
- Boolean(∗ **isSuccess**)(struct eap_sm ∗sm, void ∗priv)
- void(∗ free)(struct eap_method ∗method)

  Free EAP method data.

- int version

  Version of the EAP server method interface.

- eap_method ∗ next

  Pointer to the next EAP method.

- u8 ∗(∗ get_emsk)(struct eap_sm ∗sm, void ∗priv, size_t ∗len)

  Get EAP method specific keying extended material (EMSK).

5.1.1 Detailed Description

EAP method interface.

This structure defines the EAP method interface. Each method will need to register its own EAP type, EAP name, and set of function pointers for method specific operations. This interface is based on section 5.4 of RFC 4137.

Definition at line 30 of file eap_i.h.

5.1.2 Field Documentation

5.1.2.1 void(∗ eap_method::free)(struct eap_method ∗method)

Free EAP method data.

Parameters:

method Pointer to the method data registered with eap_server_method_register().

This function will be called when the EAP method is being unregistered. If the EAP method allocated resources during registration (e.g., allocated struct eap_method), they should be freed in this function. No other method functions will be called after this call. If this function is not defined (i.e., function pointer is NULL), a default handler is used to release the method data with free(method). This is suitable for most cases.

5.1.2.2 u8∗(∗ eap_method::get_emsk)(struct eap_sm ∗sm, void ∗priv, size_t ∗len)

Get EAP method specific keying extended material (EMSK).

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

priv Pointer to private EAP method data from eap_method::init()

len Pointer to a variable to store EMSK length

Returns:

EMSK or NULL if not available

This function can be used to get the extended keying material from the EAP method. The key may already be stored in the method-specific private data or this function may derive the key.

5.1.2.3 struct eap_method∗ eap_method::next

Pointer to the next EAP method.

This variable is used internally in the EAP method registration code to create a linked list of registered EAP methods.

Definition at line 87 of file eap_i.h.

5.1.2.4 int eap_method::version

Version of the EAP server method interface.

The EAP server method implementation should set this variable to EAP_SERVER_METHOD_-INTERFACE_VERSION. This is used to verify that the EAP method is using supported API version when using dynamically loadable EAP methods.

Definition at line 78 of file eap_i.h.

The documentation for this struct was generated from the following file:

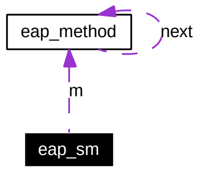
- eap_i.h

5.2 eap_sm Struct Reference

EAP server state machine data.

`#include <eap_i.h>`

Collaboration diagram for eap_sm:



Public Types

- enum {
  EAP_DISABLED, **EAP_INITIALIZE**, **EAP_IDLE**, **EAP_RECEIVED**,
  EAP_INTEGRITY_CHECK, **EAP_METHOD_RESPONSE**, **EAP_METHOD_REQUEST**, **EAP_PROPOSE_METHOD**,
  EAP_SELECT_ACTION, **EAP_SEND_REQUEST**, **EAP_DISCARD**, **EAP_NAK**,
  EAP_RETRANSMIT, **EAP_SUCCESS**, **EAP_FAILURE**, **EAP_TIMEOUT_FAILURE**,
  EAP_PICK_UP_METHOD }
- enum { **METHOD_PROPOSED**, **METHOD_CONTINUE**, **METHOD_END** }
- enum { **DECISION_SUCCESS**, **DECISION_FAILURE**, **DECISION_CONTINUE** }
- enum { **TLV_REQ_NONE**, **TLV_REQ_SUCCESS**, **TLV_REQ_FAILURE** }
- enum { **METHOD_PENDING_NONE**, **METHOD_PENDING_WAIT**, **METHOD_-PENDING_CONT** }

Data Fields

- enum eap_sm:: { ... } **EAP_state**
- int **MaxRetrans**
- u8 ∗ **eapRespData**
- size_t **eapRespDataLen**
- int **retransWhile**
- int **eapSRTT**
- int **eapRTTVAR**
- u8 ∗ **eapReqData**
- size_t **eapReqDataLen**
- u8 ∗ **eapKeyData**
- size_t **eapKeyDataLen**
- EapType **currentMethod**
- int **currentId**
- enum eap_sm:: { ... } **methodState**
- int **retransCount**
- u8 ∗ **lastReqData**
- size_t **lastReqDataLen**

- int **methodTimeout**
- Boolean **rxResp**
- int **respId**
- EapType **respMethod**
- int **respVendor**
- u32 **respVendorMethod**
- Boolean **ignore**
- enum eap_sm:: { ... } **decision**
- const struct eap_method ∗ **m**
- Boolean **changed**
- void ∗ **eapol_ctx**
- void ∗ **msg_ctx**
- eapol_callbacks ∗ **eapol_cb**
- void ∗ **eap_method_priv**
- u8 ∗ **identity**
- size_t **identity_len**
- int **lastId**
- eap_user ∗ **user**
- int **user_eap_method_index**
- int **init_phase2**
- void ∗ **ssl_ctx**
- enum eap_sm:: { ... } **tlv_request**
- void ∗ **eap_sim_db_priv**
- Boolean **backend_auth**
- Boolean **update_user**
- int **num_rounds**
- enum eap_sm:: { ... } **method_pending**

5.2.1 Detailed Description

EAP server state machine data.

Definition at line 108 of file eap_i.h.

The documentation for this struct was generated from the following file:

- eap_i.h

5.3 hostapd_bss_config Struct Reference

Per-BSS configuration.

```
#include <config.h>
```

Public Types

- enum {

  HOSTAPD_LEVEL_DEBUG_VERBOSE = 0, **HOSTAPD_LEVEL_DEBUG** = 1, **HOSTAPD_LEVEL_INFO** = 2, **HOSTAPD_LEVEL_NOTICE** = 3,

  HOSTAPD_LEVEL_WARNING = 4 }
- enum {

  HOSTAPD_DEBUG_NO = 0, **HOSTAPD_DEBUG_MINIMAL** = 1, **HOSTAPD_DEBUG_-VERBOSE** = 2, **HOSTAPD_DEBUG_MSGDUMPS** = 3,

  HOSTAPD_DEBUG_EXCESSIVE = 4 }
- enum { **ACCEPT_UNLESS_DENIED** = 0, **DENY_UNLESS_ACCEPTED** = 1, **USE_-EXTERNAL_RADIUS_AUTH** = 2 }

Data Fields

- char **iface** [IFNAMSIZ+1]
- char **bridge** [IFNAMSIZ+1]
- enum hostapd_bss_config:: { ... } **logger_syslog_level**
- enum hostapd_bss_config:: { ... } **logger_stdout_level**
- unsigned int **logger_syslog**
- unsigned int **logger_stdout**
- enum hostapd_bss_config:: { ... } **debug**
- char ∗ **dump_log_name**
- int **max_num_sta**
- int **dtim_period**
- int **ieee802_1x**
- int **eapol_version**
- int **eap_server**
- hostapd_eap_user ∗ **eap_user**
- char ∗ **eap_sim_db**
- hostapd_ip_addr **own_ip_addr**
- char ∗ **nas_identifier**
- hostapd_radius_servers ∗ **radius**
- hostapd_ssid **ssid**
- char ∗ **eap_req_id_text**
- size_t **eap_req_id_text_len**
- int **eapol_key_index_workaround**
- size_t **default_wep_key_len**
- int **individual_wep_key_len**
- int **wep_rekeying_period**
- int **broadcast_key_idx_min**
- int **broadcast_key_idx_max**
- int **eap_reauth_period**

- int **ieee802_11f**
- char **iapp_iface** [IFNAMSIZ+1]
- u8 **assoc_ap_addr** [ETH_ALEN]
- int **assoc_ap**
- enum hostapd_bss_config:: { ... } **macaddr_acl**
- macaddr ∗ **accept_mac**
- int **num_accept_mac**
- macaddr ∗ **deny_mac**
- int **num_deny_mac**
- int **auth_algs**
- int **wpa**
- int **wpa_key_mgmt**
- int **wpa_pairwise**
- int **wpa_group**
- int **wpa_group_rekey**
- int **wpa_strict_rekey**
- int **wpa_gmk_rekey**
- int **rsn_preauth**
- char ∗ **rsn_preauth_interfaces**
- int **peerkey**
- char ∗ **ctrl_interface**
- gid_t **ctrl_interface_gid**
- int **ctrl_interface_gid_set**
- char ∗ **ca_cert**
- char ∗ **server_cert**
- char ∗ **private_key**
- char ∗ **private_key_passwd**
- int **check_crl**
- char ∗ **radius_server_clients**
- int **radius_server_auth_port**
- int **radius_server_ipv6**
- char ∗ **test_socket**
- int **use_pae_group_addr**
- int **ap_max_inactivity**
- int **ignore_broadcast_ssid**
- int **wme_enabled**
- hostapd_vlan ∗ **vlan**
- hostapd_vlan ∗ **vlan_tail**
- macaddr **bssid**

5.3.1 Detailed Description

Per-BSS configuration.

Definition at line 137 of file config.h.

The documentation for this struct was generated from the following file:

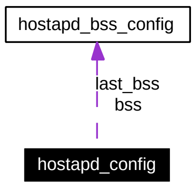
- config.h

5.4 hostapd_config Struct Reference

Per-radio interface configuration.

```
#include <config.h>
```

Collaboration diagram for hostapd_config:



Public Types

- enum { **LONG_PREAMBLE** = 0, **SHORT_PREAMBLE** = 1 }
- enum { **CTS_PROTECTION_AUTOMATIC** = 0, **CTS_PROTECTION_FORCE_ENABLED** = 1, **CTS_PROTECTION_FORCE_DISABLED** = 2, **CTS_PROTECTION_AUTOMATIC_-NO_OLBC** = 3 }
- enum { **INTERNAL_BRIDGE_DO_NOT_CONTROL** = -1, **INTERNAL_BRIDGE_-DISABLED** = 0, **INTERNAL_BRIDGE_ENABLED** = 1 }

Data Fields

- hostapd_bss_config ∗ **bss**
- hostapd_bss_config ∗ **last_bss**
- hostapd_radius_servers ∗ **radius**
- size_t **num_bss**
- u16 **beacon_int**
- int **rts_threshold**
- int **fragm_threshold**
- u8 **send_probe_response**
- u8 **channel**
- hostapd_hw_mode **hw_mode**
- enum hostapd_config:: { ... } **preamble**
- enum hostapd_config:: { ... } **cts_protection_type**
- int ∗ **supported_rates**
- int ∗ **basic_rates**
- const struct driver_ops ∗ **driver**
- int **passive_scan_interval**
- int **passive_scan_listen**
- int **passive_scan_mode**
- int **ap_table_max_size**
- int **ap_table_expiration_time**
- char **country** [3]
- int **ieee80211d**
- unsigned int **ieee80211h**
- hostapd_tx_queue_params **tx_queue** [NUM_TX_QUEUES]
- hostapd_wme_ac_params **wme_ac_params** [4]
- enum hostapd_config:: { ... } **bridge_packets**

5.4.1 Detailed Description

Per-radio interface configuration.

Definition at line 286 of file config.h.

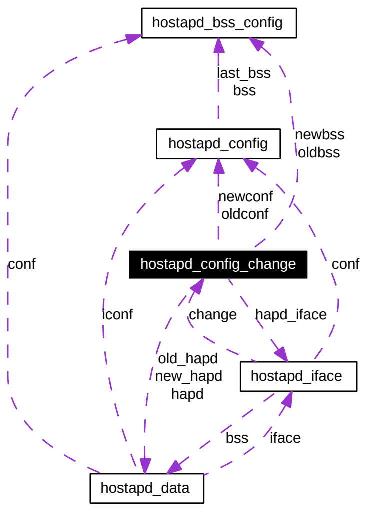
The documentation for this struct was generated from the following file:

- config.h

5.5 hostapd_config_change Struct Reference

Configuration change information.

Collaboration diagram for hostapd_config_change:



Data Fields

- hostapd_data ∗ **hapd**
- hostapd_config ∗ **newconf**
- hostapd_config ∗ **oldconf**
- hostapd_bss_config ∗ **newbss**
- hostapd_bss_config ∗ **oldbss**
- int **mac_acl_changed**
- int **num_sta_remove**
- int **beacon_changed**
- hostapd_iface ∗ **hapd_iface**
- hostapd_data ∗∗ **new_hapd**
- hostapd_data ∗∗ **old_hapd**
- int **num_old_hapd**

5.5.1 Detailed Description

Configuration change information.

This is for two purposes:

- Storing configuration information in the hostapd_iface during the asynchronous parts of reconfiguration.
- Passing configuration information for per-station reconfiguration.

Definition at line 37 of file reconfig.c.

The documentation for this struct was generated from the following file:

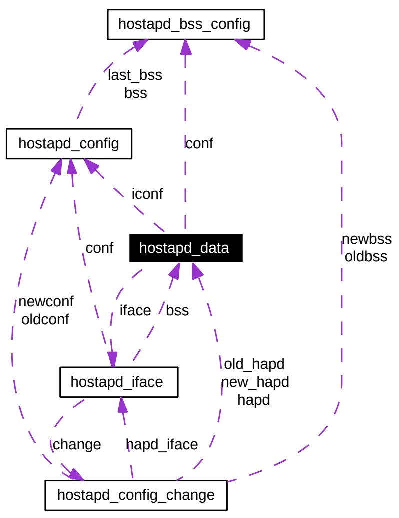
- reconfig.c

5.6 hostapd_data Struct Reference

hostapd per-BSS data structure

`#include <hostapd.h>`

Collaboration diagram for hostapd_data:



Public Types

- enum {
  DO_NOT_ASSOC = 0, **WAIT_BEACON**, **AUTHENTICATE**, **ASSOCIATE**,
  ASSOCIATED }

Data Fields

- hostapd_iface ∗ **iface**
- hostapd_config ∗ **iconf**
- hostapd_bss_config ∗ **conf**
- int **interface_added**
- u8 **own_addr** [ETH_ALEN]
- int **num_sta**
- sta_info ∗ **sta_list**
- sta_info ∗ **sta_hash** [STA_HASH_SIZE]
- sta_info ∗ **sta_aid** [MAX_AID_TABLE_SIZE]
- driver_ops ∗ **driver**
- u8 ∗ **default_wep_key**
- u8 **default_wep_key_idx**
- radius_client_data ∗ **radius**
- int **radius_client_reconfigured**
- u32 **acct_session_id_hi**
- u32 **acct_session_id_lo**
- iapp_data ∗ **iapp**

- enum hostapd_data:: { ... } **assoc_ap_state**
- char **assoc_ap_ssid** [33]
- int **assoc_ap_ssid_len**
- u16 **assoc_ap_aid**
- hostapd_cached_radius_acl ∗ **acl_cache**
- hostapd_acl_query_data ∗ **acl_queries**
- wpa_authenticator ∗ **wpa_auth**
- rsn_preauth_interface ∗ **preauth_iface**
- time_t **michael_mic_failure**
- int **michael_mic_failures**
- int **tkip_countermeasures**
- int **ctrl_sock**
- wpa_ctrl_dst ∗ **ctrl_dst**
- void ∗ **ssl_ctx**
- void ∗ **eap_sim_db_priv**
- radius_server_data ∗ **radius_srv**
- int **parameter_set_count**

5.6.1 Detailed Description

hostapd per-BSS data structure

Definition at line 114 of file hostapd.h.

The documentation for this struct was generated from the following file:

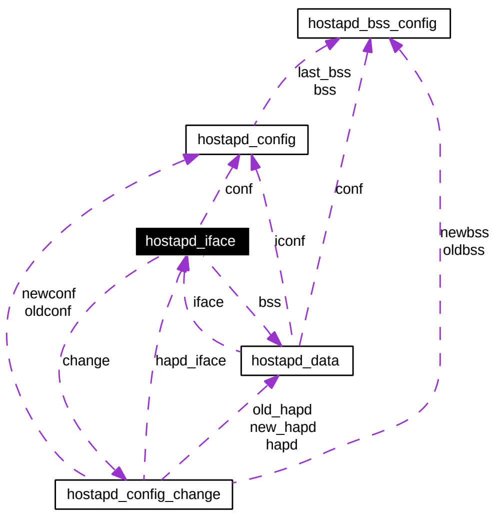
- hostapd.h

5.7 hostapd_iface Struct Reference

hostapd per-interface data structure

`#include <hostapd.h>`

Collaboration diagram for hostapd_iface:



Data Fields

- char ∗ **config_fname**
- hostapd_config ∗ **conf**
- hostapd_iface_cb **setup_cb**
- size_t **num_bss**
- hostapd_data ∗∗ **bss**
- int **num_ap**
- ap_info ∗ **ap_list**
- ap_info ∗ **ap_hash** [STA_HASH_SIZE]
- ap_info ∗ **ap_iter_list**
- hostapd_hw_modes ∗ **hw_features**
- int **num_hw_features**
- hostapd_hw_modes ∗ **current_mode**
- int **num_rates**
- hostapd_rate_data ∗ **current_rates**
- hostapd_iface_cb **hw_mode_sel_cb**
- u16 **hw_flags**
- int **num_sta_non_erp**
- int **num_sta_no_short_slot_time**
- int **num_sta_no_short_preamble**
- int **olbc**
- int **dfs_enable**
- u8 **pwr_const**
- unsigned int **tx_power**
- unsigned int **sta_max_power**

- unsigned int **channel_switch**
- hostapd_config_change ∗ **change**
- hostapd_iface_cb **reload_iface_cb**
- hostapd_iface_cb **config_reload_cb**

5.7.1 Detailed Description

hostapd per-interface data structure

Definition at line 189 of file hostapd.h.

The documentation for this struct was generated from the following file:

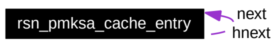
- hostapd.h

5.8 rsn_pmksa_cache_entry Struct Reference

PMKSA cache entry.

```
#include <pmksa_cache.h>
```

Collaboration diagram for rsn_pmksa_cache_entry:



Data Fields

- rsn_pmksa_cache_entry ∗ **next**
- rsn_pmksa_cache_entry ∗ **hnext**
- u8 **pmkid** [PMKID_LEN]
- u8 **pmk** [PMK_LEN]
- size_t **pmk_len**
- os_time_t **expiration**
- int **akmp**
- u8 **spa** [ETH_ALEN]
- u8 ∗ **identity**
- size_t **identity_len**
- radius_class_data **radius_class**
- u8 **eap_type_authsrv**
- int **vlan_id**

5.8.1 Detailed Description

PMKSA cache entry.

Definition at line 23 of file pmksa_cache.h.

The documentation for this struct was generated from the following file:

- pmksa_cache.h

5.9 tls_connection_params Struct Reference

Parameters for TLS connection.

`#include <tls.h>`

Data Fields

- const char ∗ **ca_cert**
- const u8 ∗ **ca_cert_blob**
- size_t **ca_cert_blob_len**
- const char ∗ **ca_path**
- const char ∗ **subject_match**
- const char ∗ **altsubject_match**
- const char ∗ **client_cert**
- const u8 ∗ **client_cert_blob**
- size_t **client_cert_blob_len**
- const char ∗ **private_key**
- const u8 ∗ **private_key_blob**
- size_t **private_key_blob_len**
- const char ∗ **private_key_passwd**
- const char ∗ **dh_file**
- const u8 ∗ **dh_blob**
- size_t **dh_blob_len**
- int **tls_ia**
- int **engine**
- const char ∗ **engine_id**
- const char ∗ **pin**
- const char ∗ **key_id**

5.9.1 Detailed Description

Parameters for TLS connection.

Parameters:

ca_cert File or reference name for CA X.509 certificate in PEM or DER format

ca_cert_blob ca_cert as inlined data or NULL if not used

ca_cert_blob_len ca_cert_blob length

ca_path Path to CA certificates (OpenSSL specific)

subject_match String to match in the subject of the peer certificate or NULL to allow all subjects

altsubject_match String to match in the alternative subject of the peer certificate or NULL to allow all alternative subjects

client_cert File or reference name for client X.509 certificate in PEM or DER format

client_cert_blob client_cert as inlined data or NULL if not used

client_cert_blob_len client_cert_blob length

private_key File or reference name for client private key in PEM or DER format (traditional format (RSA PRIVATE KEY) or PKCS#8 (PRIVATE KEY)

private_key_blob private_key as inlined data or NULL if not used

private_key_blob_len private_key_blob length

private_key_passwd Passphrase for decrypted private key, NULL if no passphrase is used.

dh_file File name for DH/DSA data in PEM format, or NULL if not used

dh_blob dh_file as inlined data or NULL if not used

dh_blob_len dh_blob length

engine 1 = use engine (e.g., a smartcard) for private key operations (this is OpenSSL specific for now)

engine_id engine id string (this is OpenSSL specific for now)

ppin pointer to the pin variable in the configuration (this is OpenSSL specific for now)

key_id the private key's key id (this is OpenSSL specific for now)

tls_ia Whether to enable TLS/IA (for EAP-TTLSv1)

TLS connection parameters to be configured with tls_connection_set_params() and tls_global_set_-params().

Certificates and private key can be configured either as a reference name (file path or reference to certificate store) or by providing the same data as a pointer to the data in memory. Only one option will be used for each field.

Definition at line 79 of file tls.h.

The documentation for this struct was generated from the following file:

- tls.h

Chapter 6

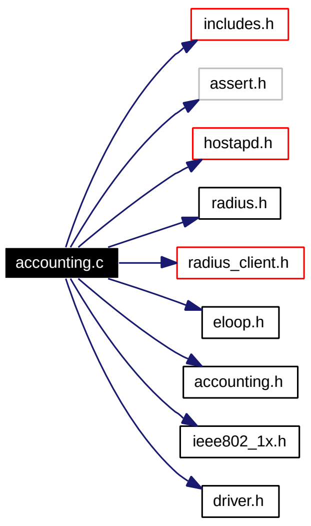
hostapd File Documentation

6.1 accounting.c File Reference

hostapd / RADIUS Accounting

```
#include "includes.h"
#include <assert.h>
#include "hostapd.h"
#include "radius.h"
#include "radius_client.h"
#include "eloop.h"
#include "accounting.h"
#include "ieee802_1x.h"
#include "driver.h"
```

Include dependency graph for accounting.c:



Defines

- #define **ACCT_DEFAULT_UPDATE_INTERVAL** 300

Functions

- const char ∗ **radius_mode_txt** (struct hostapd_data ∗hapd)
- int **radius_sta_rate** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **accounting_sta_start** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **accounting_sta_report** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int stop)
- void **accounting_sta_interim** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **accounting_sta_stop** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **accounting_sta_get_id** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- int **accounting_init** (struct hostapd_data ∗hapd)
- void **accounting_deinit** (struct hostapd_data ∗hapd)
- int **accounting_reconfig** (struct hostapd_data ∗hapd, struct hostapd_config ∗oldconf)

6.1.1 Detailed Description

hostapd / RADIUS Accounting

Copyright

Copyright (c) 2002-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

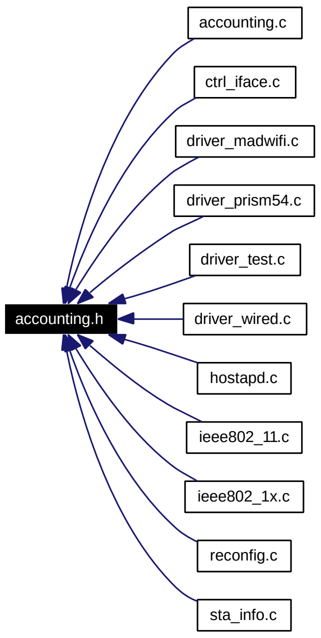
See README and COPYING for more details.

Definition in file accounting.c.

6.2 accounting.h File Reference

hostapd / RADIUS Accounting

This graph shows which files directly or indirectly include this file:



Functions

- void **accounting_sta_start** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **accounting_sta_interim** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **accounting_sta_stop** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **accounting_sta_get_id** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- int **accounting_init** (struct hostapd_data ∗hapd)
- void **accounting_deinit** (struct hostapd_data ∗hapd)
- int **accounting_reconfig** (struct hostapd_data ∗hapd, struct hostapd_config ∗oldconf)

6.2.1 Detailed Description

hostapd / RADIUS Accounting

Copyright

Copyright (c) 2002-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

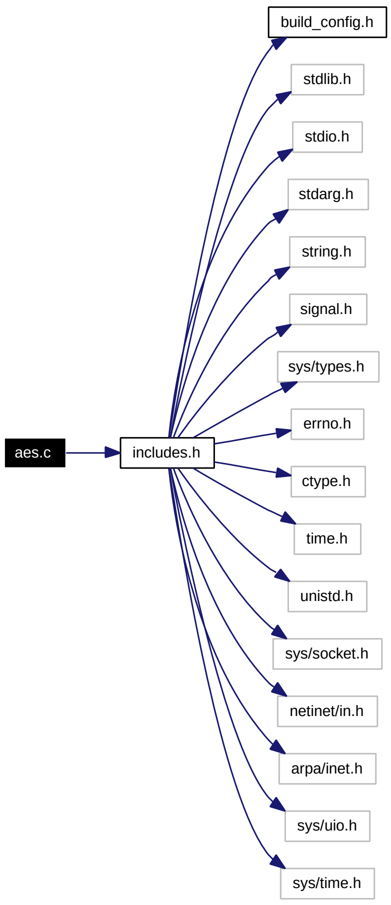
Definition in file accounting.h.

6.3 aes.c File Reference

AES (Rijndael) cipher.

`#include "includes.h"`

Include dependency graph for aes.c:



Defines

- #define **AES_SMALL_TABLES**
- #define **RCON**(i) (rcons[(i)] << 24)
- #define **TE0**(i) Te0[((i) >> 24) & 0xff]
- #define **TE1**(i) rotr(Te0[((i) >> 16) & 0xff], 8)
- #define **TE2**(i) rotr(Te0[((i) >> 8) & 0xff], 16)
- #define **TE3**(i) rotr(Te0[(i) & 0xff], 24)
- #define **TE41**(i) ((Te0[((i) >> 24) & 0xff] << 8) & 0xff000000)
- #define **TE42**(i) (Te0[((i) >> 16) & 0xff] & 0x00ff0000)
- #define **TE43**(i) (Te0[((i) >> 8) & 0xff] & 0x0000ff00)
- #define **TE44**(i) ((Te0[(i) & 0xff] >> 8) & 0x000000ff)
- #define **TE421**(i) ((Te0[((i) >> 16) & 0xff] << 8) & 0xff000000)
- #define **TE432**(i) (Te0[((i) >> 8) & 0xff] & 0x00ff0000)

- #define **TE443**(i) (Te0[(i) & 0xff] & 0x0000ff00)
- #define **TE414**(i) ((Te0[((i) >> 24) & 0xff] >> 8) & 0x000000ff)
- #define **TE4**(i) ((Te0[(i)] >> 8) & 0x000000ff)
- #define **TD0**(i) Td0[((i) >> 24) & 0xff]
- #define **TD1**(i) rotr(Td0[((i) >> 16) & 0xff], 8)
- #define **TD2**(i) rotr(Td0[((i) >> 8) & 0xff], 16)
- #define **TD3**(i) rotr(Td0[(i) & 0xff], 24)
- #define **TD41**(i) (Td4s[((i) >> 24) & 0xff] << 24)
- #define **TD42**(i) (Td4s[((i) >> 16) & 0xff] << 16)
- #define **TD43**(i) (Td4s[((i) >> 8) & 0xff] << 8)
- #define **TD44**(i) (Td4s[(i) & 0xff])
- #define **TD0_**(i) Td0[(i) & 0xff]
- #define **TD1_**(i) rotr(Td0[(i) & 0xff], 8)
- #define **TD2_**(i) rotr(Td0[(i) & 0xff], 16)
- #define **TD3_**(i) rotr(Td0[(i) & 0xff], 24)
- #define **SWAP**(x) (_lrotl(x, 8) & 0x00ff00ff | _lrotr(x, 8) & 0xff00ff00)
- #define **GETU32**(pt)
- #define **PUTU32**(ct, st)
- #define **ROUND**(i, d, s)
- #define **ROUND**(i, d, s)

Functions

- void rijndaelKeySetupEnc (u32 rk[ ], const u8 cipherKey[ ])
- void rijndaelKeySetupDec (u32 rk[ ], const u8 cipherKey[ ])
- void **rijndaelEncrypt** (const u32 rk[ ], const u8 pt[16], u8 ct[16])
- void **rijndaelDecrypt** (const u32 rk[ ], const u8 ct[16], u8 pt[16])
- void ∗ aes_encrypt_init (const u8 ∗key, size_t len)

  Initialize AES for encryption.

- void aes_encrypt (void ∗ctx, const u8 ∗plain, u8 ∗crypt)

  Encrypt one AES block.

- void aes_encrypt_deinit (void ∗ctx)

  Deinitialize AES encryption.

- void ∗ aes_decrypt_init (const u8 ∗key, size_t len)

  Initialize AES for decryption.

- void aes_decrypt (void ∗ctx, const u8 ∗crypt, u8 ∗plain)

  Decrypt one AES block.

- void aes_decrypt_deinit (void ∗ctx)

  Deinitialize AES decryption.

6.3.1 Detailed Description

AES (Rijndael) cipher.

Modifications to public domain implementation:

- support only 128-bit keys
- cleanup
- use C pre-processor to make it easier to change S table access
- added option (AES_SMALL_TABLES) for reducing code size by about 8 kB at cost of reduced throughput (quite small difference on Pentium 4, 10-25% when using -O1 or -O2 optimization)

Copyright

Copyright (c) 2003-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file aes.c.

6.3.2 Define Documentation

6.3.2.1 #define GETU32(pt)

Value:

```
(((u32)(pt)[0] << 24) ^ ((u32)(pt)[1] << 16) ^ \
((u32)(pt)[2] <<  8) ^ ((u32)(pt)[3]))
```

Definition at line 858 of file aes.c.

6.3.2.2 #define PUTU32(ct, st)

Value:

```
{ \
(ct)[0] = (u8)((st) >> 24); (ct)[1] = (u8)((st) >> 16); \
(ct)[2] = (u8)((st) >>  8); (ct)[3] = (u8)(st); }
```

Definition at line 860 of file aes.c.

6.3.2.3 #define ROUND(i, d, s)

Value:

```
d##0 = TD0(s##0) ^ TD1(s##3) ^ TD2(s##2) ^ TD3(s##1) ^ rk[4 * i]; \
d##1 = TD0(s##1) ^ TD1(s##0) ^ TD2(s##3) ^ TD3(s##2) ^ rk[4 * i + 1]; \
d##2 = TD0(s##2) ^ TD1(s##1) ^ TD2(s##0) ^ TD3(s##3) ^ rk[4 * i + 2]; \
d##3 = TD0(s##3) ^ TD1(s##2) ^ TD2(s##1) ^ TD3(s##0) ^ rk[4 * i + 3]
```

6.3.2.4 #define ROUND(i, d, s)

Value:

```
d##0 = TE0(s##0) ^ TE1(s##1) ^ TE2(s##2) ^ TE3(s##3) ^ rk[4 * i]; \
d##1 = TE0(s##1) ^ TE1(s##2) ^ TE2(s##3) ^ TE3(s##0) ^ rk[4 * i + 1]; \
d##2 = TE0(s##2) ^ TE1(s##3) ^ TE2(s##0) ^ TE3(s##1) ^ rk[4 * i + 2]; \
d##3 = TE0(s##3) ^ TE1(s##0) ^ TE2(s##1) ^ TE3(s##2) ^ rk[4 * i + 3]
```

6.3.3 Function Documentation

6.3.3.1 void aes_decrypt (void * *ctx*, const u8 * *crypt*, u8 * *plain*)

Decrypt one AES block.

Parameters:

ctx Context pointer from aes_encrypt_init()

crypt Encrypted data (16 bytes)

plain Buffer for the decrypted data (16 bytes)

Definition at line 1099 of file aes.c.

6.3.3.2 void aes_decrypt_deinit (void * *ctx*)

Deinitialize AES decryption.

Parameters:

ctx Context pointer from aes_encrypt_init()

Definition at line 1105 of file aes.c.

6.3.3.3 void* aes_decrypt_init (const u8 * *key*, size_t *len*)

Initialize AES for decryption.

Parameters:

key Decryption key

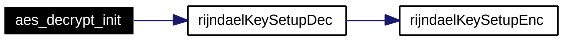
len Key length in bytes (usually 16, i.e., 128 bits)

Returns:

Pointer to context data or NULL on failure

Definition at line 1086 of file aes.c.

Here is the call graph for this function:



6.3.3.4 void aes_encrypt (void ∗ *ctx*, const u8 ∗ *plain*, u8 ∗ *crypt*)

Encrypt one AES block.

Parameters:

ctx Context pointer from aes_encrypt_init()

plain Plaintext data to be encrypted (16 bytes)

crypt Buffer for the encrypted data (16 bytes)

Definition at line 1074 of file aes.c.

6.3.3.5 void aes_encrypt_deinit (void ∗ *ctx*)

Deinitialize AES encryption.

Parameters:

ctx Context pointer from aes_encrypt_init()

Definition at line 1080 of file aes.c.

6.3.3.6 void∗ aes_encrypt_init (const u8 ∗ *key*, size_t *len*)

Initialize AES for encryption.

Parameters:

key Encryption key

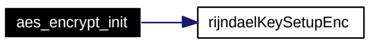
len Key length in bytes (usually 16, i.e., 128 bits)

Returns:

Pointer to context data or NULL on failure

Definition at line 1061 of file aes.c.

Here is the call graph for this function:



6.3.3.7 void rijndaelKeySetupDec (u32 *rk*[ ], const u8 *cipherKey*[ ])

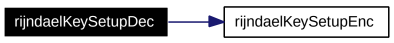
Expand the cipher key into the decryption key schedule.

Returns:

the number of rounds for the given cipher key size.

Definition at line 896 of file aes.c.

Here is the call graph for this function:



6.3.3.8 void rijndaelKeySetupEnc (u32 *rk*[ ], const u8 *cipherKey*[ ])

Expand the cipher key into the encryption key schedule.

Returns:
the number of rounds for the given cipher key size.

Definition at line 870 of file aes.c.

6.4 aes.h File Reference

AES functions.

Functions

- void ∗ **aes_encrypt_init** (const u8 ∗key, size_t len)
- void **aes_encrypt** (void ∗ctx, const u8 ∗plain, u8 ∗crypt)
- void **aes_encrypt_deinit** (void ∗ctx)
- void ∗ **aes_decrypt_init** (const u8 ∗key, size_t len)
- void **aes_decrypt** (void ∗ctx, const u8 ∗crypt, u8 ∗plain)
- void **aes_decrypt_deinit** (void ∗ctx)

6.4.1 Detailed Description

AES functions.

Definition in file aes.h.

6.5 aes_wrap.c File Reference

AES-based functions.

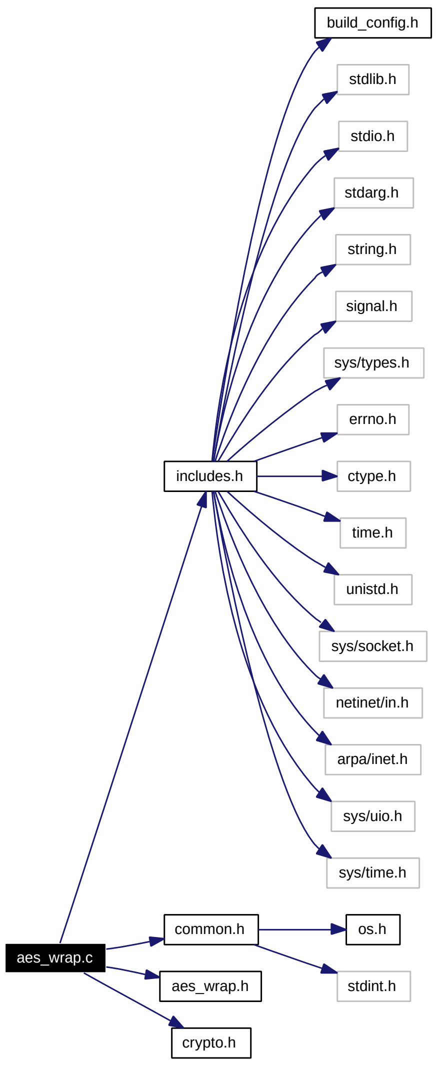
`#include "includes.h"`

`#include "common.h"`

`#include "aes_wrap.h"`

`#include "crypto.h"`

Include dependency graph for aes_wrap.c:



Defines

- #define **BLOCK_SIZE** 16

Functions

- int aes_wrap (const u8 ∗kek, int n, const u8 ∗plain, u8 ∗cipher)
  Wrap keys with AES Key Wrap Algorithm (128-bit KEK) (RFC3394).
- int aes_unwrap (const u8 ∗kek, int n, const u8 ∗cipher, u8 ∗plain)
  Unwrap key with AES Key Wrap Algorithm (128-bit KEK) (RFC3394).
- int omac1_aes_128 (const u8 ∗key, const u8 ∗data, size_t data_len, u8 ∗mac)
  One-Key CBC MAC (OMAC1) hash with AES-128 (aka AES-CMAC).
- int aes_128_encrypt_block (const u8 ∗key, const u8 ∗in, u8 ∗out)
  Perform one AES 128-bit block operation.
- int aes_128_ctr_encrypt (const u8 ∗key, const u8 ∗nonce, u8 ∗data, size_t data_len)
  AES-128 CTR mode encryption.
- int aes_128_eax_encrypt (const u8 ∗key, const u8 ∗nonce, size_t nonce_len, const u8 ∗hdr, size_t hdr_len, u8 ∗data, size_t data_len, u8 ∗tag)
  AES-128 EAX mode encryption.
- int aes_128_eax_decrypt (const u8 ∗key, const u8 ∗nonce, size_t nonce_len, const u8 ∗hdr, size_t hdr_len, u8 ∗data, size_t data_len, const u8 ∗tag)
  AES-128 EAX mode decryption.
- int aes_128_cbc_encrypt (const u8 ∗key, const u8 ∗iv, u8 ∗data, size_t data_len)
  AES-128 CBC encryption.
- int aes_128_cbc_decrypt (const u8 ∗key, const u8 ∗iv, u8 ∗data, size_t data_len)
  AES-128 CBC decryption.

6.5.1 Detailed Description

AES-based functions.

- AES Key Wrap Algorithm (128-bit KEK) (RFC3394)
  - One-Key CBC MAC (OMAC1) hash with AES-128
  - AES-128 CTR mode encryption
  - AES-128 EAX mode encryption/decryption
  - AES-128 CBC

Definition in file aes_wrap.c.

6.5.2 Function Documentation

6.5.2.1 int aes_128_cbc_decrypt (const u8 ∗ *key*, const u8 ∗ *iv*, u8 ∗ *data*, size_t *data_len*)

AES-128 CBC decryption.

Parameters:

key Decryption key

iv Decryption IV for CBC mode (16 bytes)

data Data to decrypt in-place

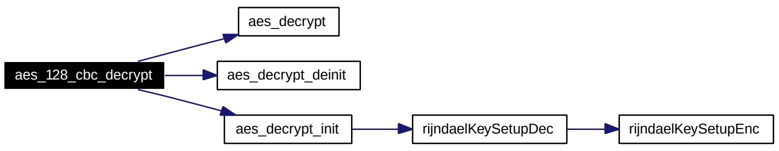
data_len Length of data in bytes (must be divisible by 16)

Returns:

0 on success, -1 on failure

Definition at line 457 of file aes_wrap.c.

Here is the call graph for this function:



6.5.2.2 int aes_128_cbc_encrypt (const u8 ∗ *key*, const u8 ∗ *iv*, u8 ∗ *data*, size_t *data_len*)

AES-128 CBC encryption.

Parameters:

key Encryption key

iv Encryption IV for CBC mode (16 bytes)

data Data to encrypt in-place

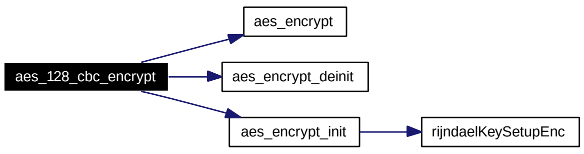
data_len Length of data in bytes (must be divisible by 16)

Returns:

0 on success, -1 on failure

Definition at line 423 of file aes_wrap.c.

Here is the call graph for this function:



6.5.2.3 int aes_128_ctr_encrypt (const u8 ∗ *key*, const u8 ∗ *nonce*, u8 ∗ *data*, size_t *data_len*)

AES-128 CTR mode encryption.

Parameters:

key Key for encryption (16 bytes)

nonce Nonce for counter mode (16 bytes)

data Data to encrypt in-place

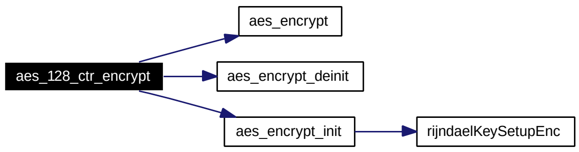
data_len Length of data in bytes

Returns:

0 on success, -1 on failure

Definition at line 253 of file aes_wrap.c.

Here is the call graph for this function:



6.5.2.4 int aes_128_eax_decrypt (const u8 ∗ *key*, const u8 ∗ *nonce*, size_t *nonce_len*, const u8 ∗ *hdr*, size_t *hdr_len*, u8 ∗ *data*, size_t *data_len*, const u8 ∗ *tag*)

AES-128 EAX mode decryption.

Parameters:

key Key for decryption (16 bytes)

nonce Nonce for counter mode

nonce_len Nonce length in bytes

hdr Header data to be authenticity protected

hdr_len Length of the header data bytes

data Data to encrypt in-place

data_len Length of data in bytes

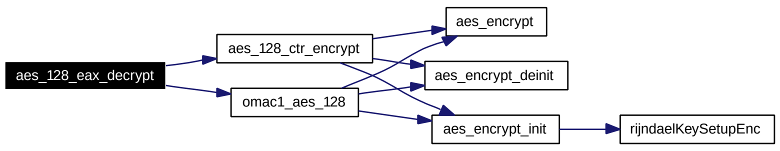
tag 16-byte tag value

Returns:

0 on success, -1 on failure, -2 if tag does not match

Definition at line 362 of file aes_wrap.c.

Here is the call graph for this function:



6.5.2.5 int aes_128_eax_encrypt (const u8 ∗ *key*, const u8 ∗ *nonce*, size_t *nonce_len*, const u8 ∗ *hdr*, size_t *hdr_len*, u8 ∗ *data*, size_t *data_len*, u8 ∗ *tag*)

AES-128 EAX mode encryption.

Parameters:

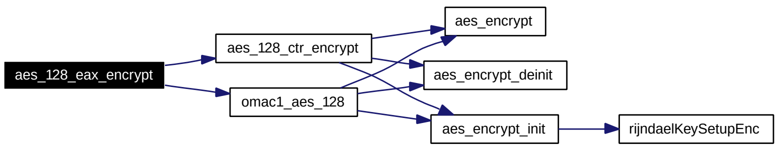
key	Key for encryption (16 bytes)
nonce	Nonce for counter mode
nonce_len	Nonce length in bytes
hdr	Header data to be authenticity protected
hdr_len	Length of the header data bytes
data	Data to encrypt in-place
data_len	Length of data in bytes
tag	16-byte tag value

Returns:

0 on success, -1 on failure

Definition at line 304 of file aes_wrap.c.

Here is the call graph for this function:



6.5.2.6 int aes_128_encrypt_block (const u8 ∗ *key*, const u8 ∗ *in*, u8 ∗ *out*)

Perform one AES 128-bit block operation.

Parameters:

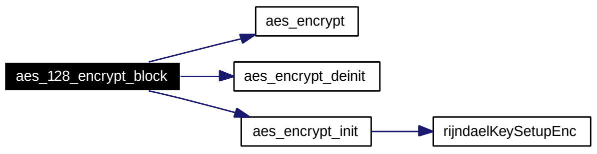
key	Key for AES
in	Input data (16 bytes)
out	Output of the AES block operation (16 bytes)

Returns:

0 on success, -1 on failure

Definition at line 230 of file aes_wrap.c.

Here is the call graph for this function:



6.5.2.7 int aes_unwrap (const u8 ∗ *kek*, int *n*, const u8 ∗ *cipher*, u8 ∗ *plain*)

Unwrap key with AES Key Wrap Algorithm (128-bit KEK) (RFC3394).

Parameters:

kek Key encryption key (KEK)

n Length of the wrapped key in 64-bit units; e.g., 2 = 128-bit = 16 bytes

cipher Wrapped key to be unwrapped, (n + 1) ∗ 64 bit

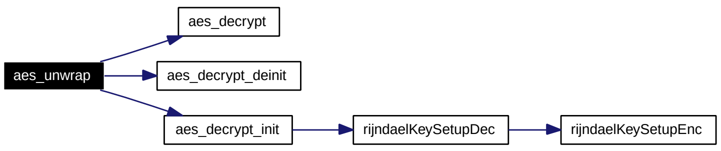
plain Plaintext key, n ∗ 64 bit

Returns:

0 on success, -1 on failure (e.g., integrity verification failed)

Definition at line 104 of file aes_wrap.c.

Here is the call graph for this function:



6.5.2.8 int aes_wrap (const u8 ∗ *kek*, int *n*, const u8 ∗ *plain*, u8 ∗ *cipher*)

Wrap keys with AES Key Wrap Algorithm (128-bit KEK) (RFC3394).

Parameters:

kek Key encryption key (KEK)

n Length of the wrapped key in 64-bit units; e.g., 2 = 128-bit = 16 bytes

plain Plaintext key to be wrapped, n ∗ 64 bit

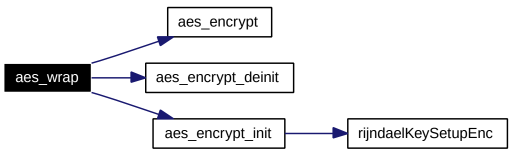
cipher Wrapped key, (n + 1) ∗ 64 bit

Returns:

0 on success, -1 on failure

Definition at line 45 of file aes_wrap.c.

Here is the call graph for this function:



6.5.2.9 int omac1_aes_128 (const u8 $*$ *key*, const u8 $*$ *data*, size_t *data_len*, u8 $*$ *mac*)

One-Key CBC MAC (OMAC1) hash with AES-128 (aka AES-CMAC).

Parameters:

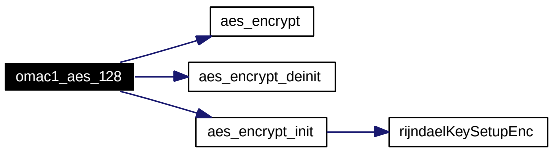
key	128-bit key for the hash operation
data	Data buffer for which a MAC is determined
data	Length of data buffer in bytes
mac	Buffer for MAC (128 bits, i.e., 16 bytes)

Returns:

0 on success, -1 on failure

Definition at line 181 of file aes_wrap.c.

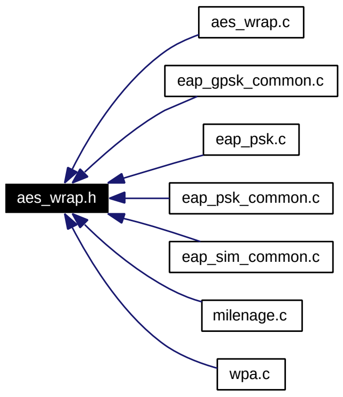
Here is the call graph for this function:



6.6 aes_wrap.h File Reference

AES-based functions.

This graph shows which files directly or indirectly include this file:



Functions

- int aes_wrap (const u8 ∗kek, int n, const u8 ∗plain, u8 ∗cipher)

  Wrap keys with AES Key Wrap Algorithm (128-bit KEK) (RFC3394).

- int aes_unwrap (const u8 ∗kek, int n, const u8 ∗cipher, u8 ∗plain)

  Unwrap key with AES Key Wrap Algorithm (128-bit KEK) (RFC3394).

- int omac1_aes_128 (const u8 ∗key, const u8 ∗data, size_t data_len, u8 ∗mac)

  One-Key CBC MAC (OMAC1) hash with AES-128 (aka AES-CMAC).

- int aes_128_encrypt_block (const u8 ∗key, const u8 ∗in, u8 ∗out)

  Perform one AES 128-bit block operation.

- int aes_128_ctr_encrypt (const u8 ∗key, const u8 ∗nonce, u8 ∗data, size_t data_len)

  AES-128 CTR mode encryption.

- int aes_128_eax_encrypt (const u8 ∗key, const u8 ∗nonce, size_t nonce_len, const u8 ∗hdr, size_t hdr_len, u8 ∗data, size_t data_len, u8 ∗tag)

  AES-128 EAX mode encryption.

- int aes_128_eax_decrypt (const u8 ∗key, const u8 ∗nonce, size_t nonce_len, const u8 ∗hdr, size_t hdr_len, u8 ∗data, size_t data_len, const u8 ∗tag)

  AES-128 EAX mode decryption.

- int aes_128_cbc_encrypt (const u8 ∗key, const u8 ∗iv, u8 ∗data, size_t data_len)

  AES-128 CBC encryption.

- int aes_128_cbc_decrypt (const u8 ∗key, const u8 ∗iv, u8 ∗data, size_t data_len)

  AES-128 CBC decryption.

6.6.1 Detailed Description

AES-based functions.

- AES Key Wrap Algorithm (128-bit KEK) (RFC3394)
  - One-Key CBC MAC (OMAC1) hash with AES-128
  - AES-128 CTR mode encryption
  - AES-128 EAX mode encryption/decryption
  - AES-128 CBC

Copyright

Copyright (c) 2003-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file aes_wrap.h.

6.6.2 Function Documentation

6.6.2.1 int aes_128_cbc_decrypt (const u8 ∗ *key*, const u8 ∗ *iv*, u8 ∗ *data*, size_t *data_len*)

AES-128 CBC decryption.

Parameters:

key Decryption key

iv Decryption IV for CBC mode (16 bytes)

data Data to decrypt in-place

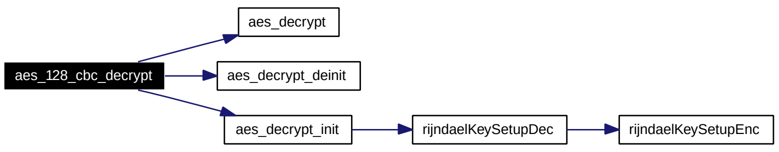
data_len Length of data in bytes (must be divisible by 16)

Returns:

0 on success, -1 on failure

Definition at line 457 of file aes_wrap.c.

Here is the call graph for this function:



6.6.2.2 int aes_128_cbc_encrypt (const u8 ∗ *key*, const u8 ∗ *iv*, u8 ∗ *data*, size_t *data_len*)

AES-128 CBC encryption.

Parameters:

key Encryption key

iv Encryption IV for CBC mode (16 bytes)

data Data to encrypt in-place

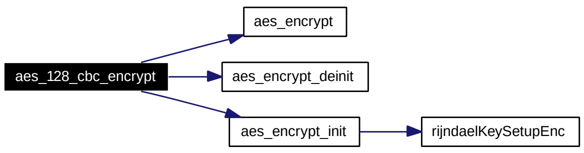
data_len Length of data in bytes (must be divisible by 16)

Returns:

0 on success, -1 on failure

Definition at line 423 of file aes_wrap.c.

Here is the call graph for this function:



6.6.2.3 int aes_128_ctr_encrypt (const u8 ∗ *key*, const u8 ∗ *nonce*, u8 ∗ *data*, size_t *data_len*)

AES-128 CTR mode encryption.

Parameters:

key Key for encryption (16 bytes)

nonce Nonce for counter mode (16 bytes)

data Data to encrypt in-place

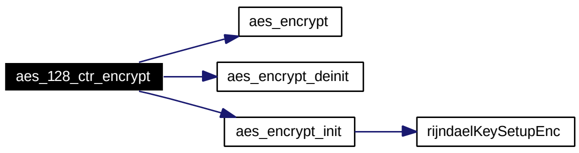
data_len Length of data in bytes

Returns:

0 on success, -1 on failure

Definition at line 253 of file aes_wrap.c.

Here is the call graph for this function:



6.6.2.4 int aes_128_eax_decrypt (const u8 ∗ *key*, const u8 ∗ *nonce*, size_t *nonce_len*, const u8 ∗ *hdr*, size_t *hdr_len*, u8 ∗ *data*, size_t *data_len*, const u8 ∗ *tag*)

AES-128 EAX mode decryption.

Parameters:

key Key for decryption (16 bytes)

nonce Nonce for counter mode

nonce_len Nonce length in bytes

hdr Header data to be authenticity protected

hdr_len Length of the header data bytes

data Data to encrypt in-place

data_len Length of data in bytes

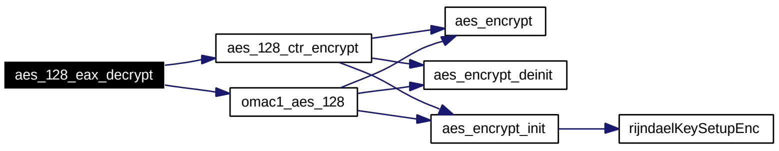
tag 16-byte tag value

Returns:

0 on success, -1 on failure, -2 if tag does not match

Definition at line 362 of file aes_wrap.c.

Here is the call graph for this function:



6.6.2.5 int aes_128_eax_encrypt (const u8 ∗ *key*, const u8 ∗ *nonce*, size_t *nonce_len*, const u8 ∗ *hdr*, size_t *hdr_len*, u8 ∗ *data*, size_t *data_len*, u8 ∗ *tag*)

AES-128 EAX mode encryption.

Parameters:

key Key for encryption (16 bytes)

nonce Nonce for counter mode

nonce_len Nonce length in bytes

hdr Header data to be authenticity protected

hdr_len Length of the header data bytes

data Data to encrypt in-place

data_len Length of data in bytes

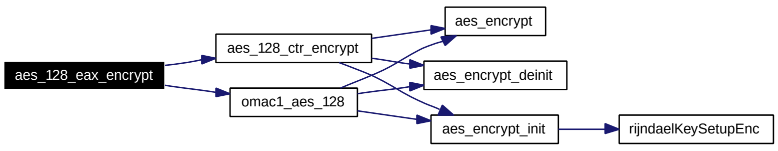
tag 16-byte tag value

Returns:

0 on success, -1 on failure

Definition at line 304 of file aes_wrap.c.

Here is the call graph for this function:



6.6.2.6 int aes_128_encrypt_block (const u8 ∗ *key*, const u8 ∗ *in*, u8 ∗ *out*)

Perform one AES 128-bit block operation.

Parameters:

key Key for AES

in Input data (16 bytes)

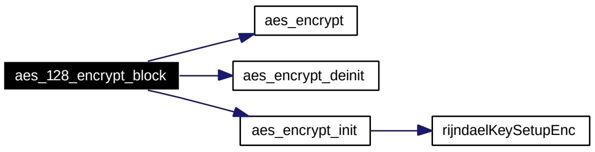
out Output of the AES block operation (16 bytes)

Returns:

0 on success, -1 on failure

Definition at line 230 of file aes_wrap.c.

Here is the call graph for this function:



6.6.2.7 int aes_unwrap (const u8 ∗ *kek*, int *n*, const u8 ∗ *cipher*, u8 ∗ *plain*)

Unwrap key with AES Key Wrap Algorithm (128-bit KEK) (RFC3394).

Parameters:

kek Key encryption key (KEK)

n Length of the wrapped key in 64-bit units; e.g., 2 = 128-bit = 16 bytes

cipher Wrapped key to be unwrapped, (n + 1) ∗ 64 bit

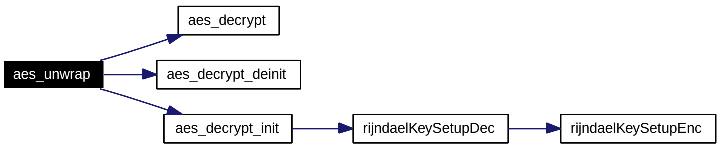
plain Plaintext key, n ∗ 64 bit

Returns:

0 on success, -1 on failure (e.g., integrity verification failed)

Definition at line 104 of file aes_wrap.c.

Here is the call graph for this function:



6.6.2.8 int aes_wrap (const u8 $*$ *kek*, int *n*, const u8 $*$ *plain*, u8 $*$ *cipher*)

Wrap keys with AES Key Wrap Algorithm (128-bit KEK) (RFC3394).

Parameters:

kek Key encryption key (KEK)

n Length of the wrapped key in 64-bit units; e.g., 2 = 128-bit = 16 bytes

plain Plaintext key to be wrapped, n $*$ 64 bit

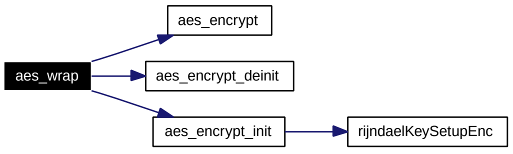
cipher Wrapped key, (n + 1) $*$ 64 bit

Returns:

0 on success, -1 on failure

Definition at line 45 of file aes_wrap.c.

Here is the call graph for this function:



6.6.2.9 int omac1_aes_128 (const u8 $*$ *key*, const u8 $*$ *data*, size_t *data_len*, u8 $*$ *mac*)

One-Key CBC MAC (OMAC1) hash with AES-128 (aka AES-CMAC).

Parameters:

key 128-bit key for the hash operation

data Data buffer for which a MAC is determined

data Length of data buffer in bytes

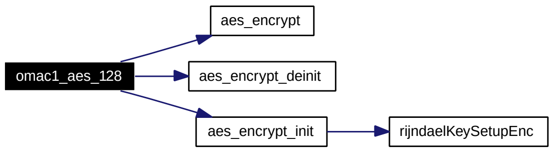
mac Buffer for MAC (128 bits, i.e., 16 bytes)

Returns:

0 on success, -1 on failure

Definition at line 181 of file aes_wrap.c.

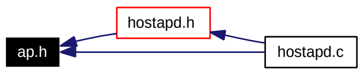
Here is the call graph for this function:


aes_encrypt
omac1_aes_128
aes_encrypt_deinit
aes_encrypt_init
rijndaelKeySetupEnc

6.7 ap.h File Reference

hostapd / Station table data structures

This graph shows which files directly or indirectly include this file:



Defines

- #define **WLAN_STA_AUTH** BIT(0)
- #define **WLAN_STA_ASSOC** BIT(1)
- #define **WLAN_STA_PS** BIT(2)
- #define **WLAN_STA_TIM** BIT(3)
- #define **WLAN_STA_PERM** BIT(4)
- #define **WLAN_STA_AUTHORIZED** BIT(5)
- #define **WLAN_STA_PENDING_POLL** BIT(6)
- #define **WLAN_STA_SHORT_PREAMBLE** BIT(7)
- #define **WLAN_STA_PREAUTH** BIT(8)
- #define **WLAN_STA_WME** BIT(9)
- #define **WLAN_STA_NONERP** BIT(31)
- #define **WLAN_SUPP_RATES_MAX** 32
- #define **MAX_AID_TABLE_SIZE** 128
- #define **STA_HASH_SIZE** 256
- #define **STA_HASH**(sta) (sta[5])
- #define **AP_MAX_INACTIVITY** (5 ∗ 60)
- #define **AP_DISASSOC_DELAY** (1)
- #define **AP_DEAUTH_DELAY** (1)
- #define **AP_MAX_INACTIVITY_AFTER_DISASSOC** (1 ∗ 30)
- #define **AP_MAX_INACTIVITY_AFTER_DEAUTH** (1 ∗ 5)

6.7.1 Detailed Description

hostapd / Station table data structures

Copyright

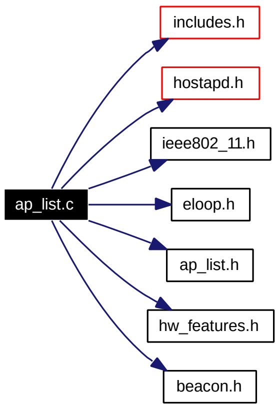
Definition in file ap.h.

6.8 ap_list.c File Reference

hostapd / AP table

```
#include "includes.h"
#include "hostapd.h"
#include "ieee802_11.h"
#include "eloop.h"
#include "ap_list.h"
#include "hw_features.h"
#include "beacon.h"
```

Include dependency graph for ap_list.c:



Enumerations

- enum **ieee80211_phytype** {

  ieee80211_phytype_fhss_dot11_97 = 1, **ieee80211_phytype_dsss_dot11_97** = 2, **ieee80211_-phytype_irbaseband** = 3, **ieee80211_phytype_dsss_dot11_b** = 4,

  ieee80211_phytype_pbcc_dot11_b = 5, **ieee80211_phytype_ofdm_dot11_g** = 6, **ieee80211_-phytype_pbcc_dot11_g** = 7, **ieee80211_phytype_ofdm_dot11_a** = 8,

  ieee80211_phytype_dsss_dot11_turbog = 255, **ieee80211_phytype_dsss_dot11_turbo** = 256 }

Functions

- ap_info ∗ **ap_get_ap** (struct hostapd_iface ∗iface, u8 ∗ap)
- int **ap_ap_for_each** (struct hostapd_iface ∗iface, int(∗func)(struct ap_info ∗s, void ∗data), void ∗data)
- void **ap_list_process_beacon** (struct hostapd_iface ∗iface, struct ieee80211_mgmt ∗mgmt, struct ieee802_11_elems ∗elems, struct hostapd_frame_info ∗fi)
- int **ap_list_init** (struct hostapd_iface ∗iface)
- void **ap_list_deinit** (struct hostapd_iface ∗iface)
- int **ap_list_reconfig** (struct hostapd_iface ∗iface, struct hostapd_config ∗oldconf)

Variables

- ieee80211_frame_info **packed**

6.8.1 Detailed Description

hostapd / AP table

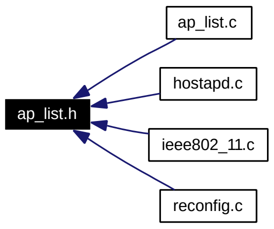
Copyright

Definition in file ap_list.c.

6.9 ap_list.h File Reference

hostapd / AP table

This graph shows which files directly or indirectly include this file:



Functions

- ap_info ∗ **ap_get_ap** (struct hostapd_iface ∗iface, u8 ∗sta)
- int **ap_ap_for_each** (struct hostapd_iface ∗iface, int(∗func)(struct ap_info ∗s, void ∗data), void ∗data)
- void **ap_list_process_beacon** (struct hostapd_iface ∗iface, struct ieee80211_mgmt ∗mgmt, struct ieee802_11_elems ∗elems, struct hostapd_frame_info ∗fi)
- int **ap_list_init** (struct hostapd_iface ∗iface)
- void **ap_list_deinit** (struct hostapd_iface ∗iface)
- int **ap_list_reconfig** (struct hostapd_iface ∗iface, struct hostapd_config ∗oldconf)

6.9.1 Detailed Description

hostapd / AP table

Copyright

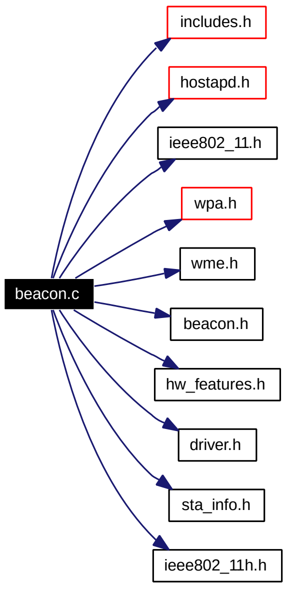
Definition in file ap_list.h.

6.10 beacon.c File Reference

hostapd / IEEE 802.11 Management: Beacon and Probe Request/Response

```
#include "includes.h"
#include "hostapd.h"
#include "ieee802_11.h"
#include "wpa.h"
#include "wme.h"
#include "beacon.h"
#include "hw_features.h"
#include "driver.h"
#include "sta_info.h"
#include "ieee802_11h.h"
```

Include dependency graph for beacon.c:



Defines

- #define **MAX_PROBERESP_LEN** 512
- #define **BEACON_HEAD_BUF_SIZE** 256
- #define **BEACON_TAIL_BUF_SIZE** 256

Functions

- void **handle_probe_req** (struct hostapd_data ∗hapd, struct ieee80211_mgmt ∗mgmt, size_t len)
- void **ieee802_11_set_beacon** (struct hostapd_data ∗hapd)
- void **ieee802_11_set_beacons** (struct hostapd_iface ∗iface)

6.10.1 Detailed Description

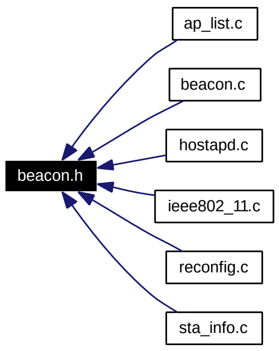
hostapd / IEEE 802.11 Management: Beacon and Probe Request/Response

Definition in file beacon.c.

6.11 beacon.h File Reference

hostapd / IEEE 802.11 Management: Beacon and Probe Request/Response

This graph shows which files directly or indirectly include this file:



Functions

- void **handle_probe_req** (struct hostapd_data ∗hapd, struct ieee80211_mgmt ∗mgmt, size_t len)
- void **ieee802_11_set_beacon** (struct hostapd_data ∗hapd)
- void **ieee802_11_set_beacons** (struct hostapd_iface ∗iface)

6.11.1 Detailed Description

hostapd / IEEE 802.11 Management: Beacon and Probe Request/Response

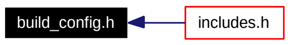
Copyright

Definition in file beacon.h.

6.12 build_config.h File Reference

wpa_supplicant/hostapd - Build time configuration defines

This graph shows which files directly or indirectly include this file:



6.12.1 Detailed Description

wpa_supplicant/hostapd - Build time configuration defines

Copyright

Copyright (c) 2005-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

This header file can be used to define configuration defines that were originally defined in Makefile. This is mainly meant for IDE use or for systems that do not have suitable 'make' tool. In these cases, it may be easier to have a single place for defining all the needed C pre-processor defines.

Definition in file build_config.h.

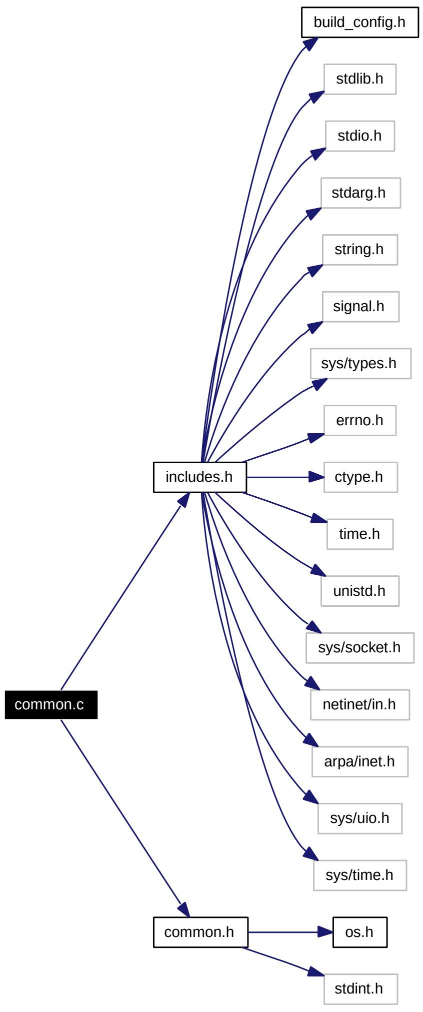
6.13 common.c File Reference

wpa_supplicant/hostapd / common helper functions, etc.

```
#include "includes.h"
```

```
#include "common.h"
```

Include dependency graph for common.c:



Functions

- int hwaddr_aton (const char ∗txt, u8 ∗addr)

  Convert ASCII string to MAC address.

- int hexstr2bin (const char ∗hex, u8 ∗buf, size_t len)

  Convert ASCII hex string into binary data.

- void inc_byte_array (u8 ∗counter, size_t len)

  Increment arbitrary length byte array by one.

- void **wpa_get_ntp_timestamp** (u8 ∗buf)
- void wpa_debug_print_timestamp (void)

  Print timestamp for debug output.

- void wpa_printf (int level, char ∗fmt,...)

  conditional printf

- void wpa_hexdump (int level, const char ∗title, const u8 ∗buf, size_t len)

  conditional hex dump

- void wpa_hexdump_key (int level, const char ∗title, const u8 ∗buf, size_t len)

  conditional hex dump, hide keys

- void wpa_hexdump_ascii (int level, const char ∗title, const u8 ∗buf, size_t len)

  conditional hex dump

- void wpa_hexdump_ascii_key (int level, const char ∗title, const u8 ∗buf, size_t len)

  conditional hex dump, hide keys

- int **wpa_debug_open_file** (void)
- void **wpa_debug_close_file** (void)
- void wpa_msg_register_cb (wpa_msg_cb_func func)

  Register callback function for wpa_msg() messages.

- void **wpa_msg** (void ∗ctx, int level, char ∗fmt,...)
- int wpa_snprintf_hex (char ∗buf, size_t buf_size, const u8 ∗data, size_t len)

  Print data as a hex string into a buffer.

- int wpa_snprintf_hex_uppercase (char ∗buf, size_t buf_size, const u8 ∗data, size_t len)

  Print data as a upper case hex string into buf.

- const char ∗ wpa_ssid_txt (u8 ∗ssid, size_t ssid_len)

  Convert SSID to a printable string.

Variables

- int **wpa_debug_use_file** = 0
- int **wpa_debug_level** = MSG_INFO
- int **wpa_debug_show_keys** = 0
- int **wpa_debug_timestamp** = 0

6.13.1 Detailed Description

wpa_supplicant/hostapd / common helper functions, etc.

Copyright

Copyright (c) 2002-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

Definition in file common.c.

6.13.2 Function Documentation

6.13.2.1 int hexstr2bin (const char ∗ *hex*, u8 ∗ *buf*, size_t *len*)

Convert ASCII hex string into binary data.

Parameters:

hex ASCII hex string (e.g., "01ab")

buf Buffer for the binary data

len Length of the text to convert in bytes (of buf); hex will be double this size

Returns:

0 on success, -1 on failure (invalid hex string)

Definition at line 93 of file common.c.

6.13.2.2 int hwaddr_aton (const char ∗ *txt*, u8 ∗ *addr*)

Convert ASCII string to MAC address.

Parameters:

txt MAC address as a string (e.g., "00:11:22:33:44:55")

addr Buffer for the MAC address (ETH_ALEN = 6 bytes)

Returns:

0 on success, -1 on failure (e.g., string not a MAC address)

Definition at line 62 of file common.c.

6.13.2.3 void inc_byte_array (u8 ∗ *counter*, size_t *len*)

Increment arbitrary length byte array by one.

Parameters:

counter Pointer to byte array

len Length of the counter in bytes

This function increments the last byte of the counter by one and continues rolling over to more significant bytes if the byte was incremented from 0xff to 0x00.

Definition at line 121 of file common.c.

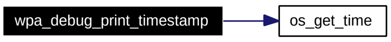
6.13.2.4 void wpa_debug_print_timestamp (void)

Print timestamp for debug output.

This function prints a timestamp in <seconds from 1970>.<microsoconds> format if debug output has been configured to include timestamps in debug messages.

Definition at line 152 of file common.c.

Here is the call graph for this function:



6.13.2.5 void wpa_hexdump (int *level*, const char ∗ *title*, const u8 ∗ *buf*, size_t *len*)

conditional hex dump

Parameters:

level priority level (MSG_∗) of the message

title title of for the message

buf data buffer to be dumped

len length of the buf

This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration. The contents of buf is printed out has hex dump.

Definition at line 242 of file common.c.

6.13.2.6 void wpa_hexdump_ascii (int *level*, const char ∗ *title*, const u8 ∗ *buf*, size_t *len*)

conditional hex dump

Parameters:

level priority level (MSG_∗) of the message

title title of for the message

buf data buffer to be dumped

len length of the buf

This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration. The contents of buf is printed out has hex dump with both the hex numbers and ASCII characters (for printable range) are shown. 16 bytes per line will be shown.

Definition at line 339 of file common.c.

6.13.2.7 void wpa_hexdump_ascii_key (int *level*, const char ∗ *title*, const u8 ∗ *buf*, size_t *len*)

conditional hex dump, hide keys

Parameters:

level priority level (MSG_∗) of the message

title title of for the message

buf data buffer to be dumped

len length of the buf

This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration. The contents of buf is printed out has hex dump with both the hex numbers and ASCII characters (for printable range) are shown. 16 bytes per line will be shown. This works like wpa_hexdump_ascii(), but by default, does not include secret keys (passwords, etc.) in debug output.

Definition at line 345 of file common.c.

6.13.2.8 void wpa_hexdump_key (int *level*, const char ∗ *title*, const u8 ∗ *buf*, size_t *len*)

conditional hex dump, hide keys

Parameters:

level priority level (MSG_∗) of the message

title title of for the message

buf data buffer to be dumped

len length of the buf

This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration. The contents of buf is printed out has hex dump. This works like wpa_hexdump(), but by default, does not include secret keys (passwords, etc.) in debug output.

Definition at line 248 of file common.c.

6.13.2.9 void wpa_msg_register_cb (wpa_msg_cb_func *func*)

Register callback function for wpa_msg() messages.

Parameters:

func Callback function (NULL to unregister)

Definition at line 390 of file common.c.

6.13.2.10 void wpa_printf (int *level*, char ∗ *fmt*, ...)

conditional printf

Parameters:

level priority level (MSG_∗) of the message

fmt printf format string, followed by optional arguments

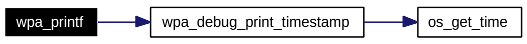
This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration.

Note: New line '

' is added to the end of the text when printing to stdout.

Definition at line 182 of file common.c.

Here is the call graph for this function:



6.13.2.11 int wpa_snprintf_hex (char ∗ *buf*, size_t *buf_size*, const u8 ∗ *data*, size_t *len*)

Print data as a hex string into a buffer.

Parameters:

buf Memory area to use as the output buffer

buf_size Maximum buffer size in bytes (should be at least 2 ∗ len + 1)

data Data to be printed

len Length of data in bytes

Returns:

Number of bytes written

Definition at line 450 of file common.c.

6.13.2.12 int wpa_snprintf_hex_uppercase (char ∗ *buf*, size_t *buf_size*, const u8 ∗ *data*, size_t *len*)

Print data as a upper case hex string into buf.

Parameters:

buf Memory area to use as the output buffer

buf_size Maximum buffer size in bytes (should be at least 2 ∗ len + 1)

data Data to be printed

len Length of data in bytes

Returns:

Number of bytes written

Definition at line 465 of file common.c.

6.13.2.13 const char∗ wpa_ssid_txt (u8 ∗ *ssid*, size_t *ssid_len*)

Convert SSID to a printable string.

Parameters:

ssid SSID (32-octet string)

ssid_len Length of ssid in octets

Returns:

Pointer to a printable string

This function can be used to convert SSIDs into printable form. In most cases, SSIDs do not use unprintable characters, but IEEE 802.11 standard does not limit the used character set, so anything could be used in an SSID.

This function uses a static buffer, so only one call can be used at the time, i.e., this is not re-entrant and the returned buffer must be used before calling this again.

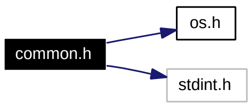
Definition at line 598 of file common.c.

6.14 common.h File Reference

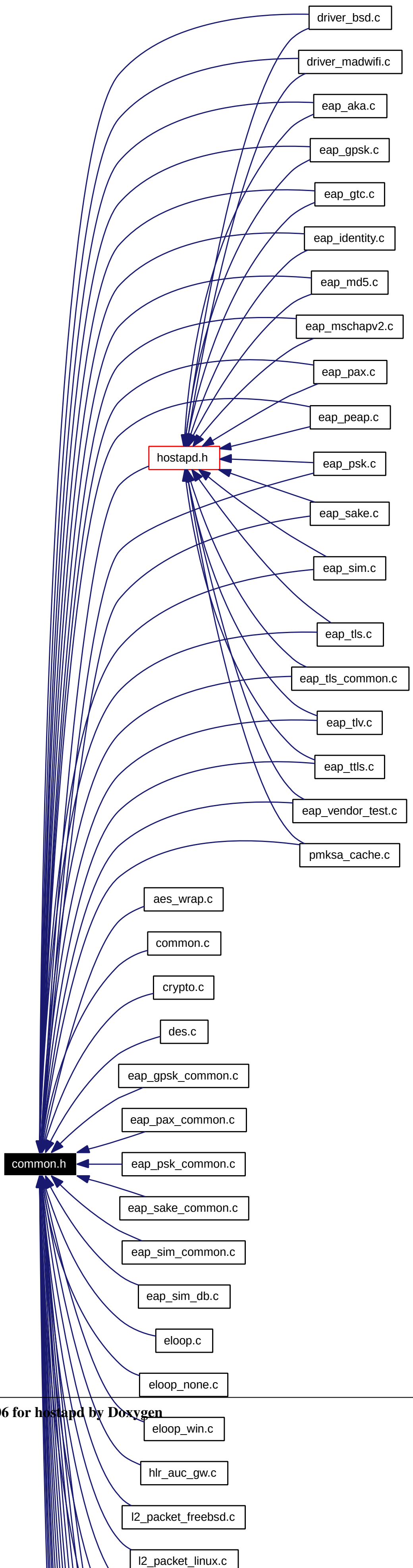
wpa_supplicant/hostapd / common helper functions, etc.

```
#include "os.h"
#include <stdint.h>
```

Include dependency graph for common.h:



This graph shows which files directly or indirectly include this file:


driver_bsd.c
driver_madwifi.c
eap_aka.c
eap_gpsk.c
eap_gtc.c
eap_identity.c
eap_md5.c
eap_mschapv2.c
eap_pax.c
eap_peap.c
hostapd.h
eap_psk.c
eap_sake.c
eap_sim.c
eap_tls.c
eap_tls_common.c
eap_tlv.c
eap_ttls.c
eap_vendor_test.c
pmksa_cache.c
aes_wrap.c
common.c
crypto.c
des.c
eap_gpsk_common.c
eap_pax_common.c
common.h
eap_psk_common.c
eap_sake_common.c
eap_sim_common.c
eap_sim_db.c
eloop.c
eloop_none.c
eloop_win.c
hlr_auc_gw.c
l2_packet_freebsd.c
l2_packet_linux.c

Defines

- #define **__LITTLE_ENDIAN** 1234
- #define **__BIG_ENDIAN** 4321
- #define **WPA_GET_BE16**(a) ((u16) (((a)[0] << 8) | (a)[1]))
- #define **WPA_PUT_BE16**(a, val)
- #define **WPA_GET_LE16**(a) ((u16) (((a)[1] << 8) | (a)[0]))
- #define **WPA_PUT_LE16**(a, val)
- #define **WPA_GET_BE24**(a)
- #define **WPA_PUT_BE24**(a, val)
- #define **WPA_GET_BE32**(a)
- #define **WPA_PUT_BE32**(a, val)
- #define **WPA_PUT_BE64**(a, val)
- #define **ETH_ALEN** 6
- #define **WPA_TYPES_DEFINED**
- #define **hostapd_get_rand** os_get_random
- #define **PRINTF_FORMAT**(a, b)
- #define **STRUCT_PACKED**
- #define **WPA_ASSERT**(a) do { } while (0)
- #define **wpa_zalloc**(s) os_zalloc((s))
- #define **wpa_unicode2ascii_inplace**(s) do { } while (0)
- #define **wpa_strdup_tchar**(s) strdup((s))

Typedefs

- typedef uint64_t **u64**
- typedef uint32_t **u32**
- typedef uint16_t **u16**
- typedef uint8_t **u8**
- typedef int64_t **s64**
- typedef int32_t **s32**
- typedef int16_t **s16**
- typedef int8_t **s8**
- typedef u32 **__be32**
- typedef u64 **__be64**

Enumerations

- enum {

  MSG_MSGDUMP, **MSG_DEBUG**, **MSG_INFO**, **MSG_WARNING**,

  MSG_ERROR }

Functions

- int hwaddr_aton (const char ∗txt, u8 ∗addr)

  Convert ASCII string to MAC address.

- int hexstr2bin (const char ∗hex, u8 ∗buf, size_t len)

  Convert ASCII hex string into binary data.

- void inc_byte_array (u8 ∗counter, size_t len)

  Increment arbitrary length byte array by one.

- void **wpa_get_ntp_timestamp** (u8 ∗buf)
- int **wpa_debug_open_file** (void)
- void **wpa_debug_close_file** (void)
- void wpa_debug_print_timestamp (void)

  Print timestamp for debug output.

- void wpa_printf (int level, char ∗fmt,...) PRINTF_FORMAT(2

  conditional printf

- void void wpa_hexdump (int level, const char ∗title, const u8 ∗buf, size_t len)

  conditional hex dump

- void wpa_hexdump_key (int level, const char ∗title, const u8 ∗buf, size_t len)

  conditional hex dump, hide keys

- void wpa_hexdump_ascii (int level, const char ∗title, const u8 ∗buf, size_t len)

  conditional hex dump

- void wpa_hexdump_ascii_key (int level, const char ∗title, const u8 ∗buf, size_t len)

  conditional hex dump, hide keys

- void wpa_msg (void ∗ctx, int level, char ∗fmt,...) PRINTF_FORMAT(3

  Conditional printf for default target and ctrl_iface monitors.

- void wpa_msg_register_cb (wpa_msg_cb_func func)

  Register callback function for wpa_msg() messages.

- int wpa_snprintf_hex (char ∗buf, size_t buf_size, const u8 ∗data, size_t len)

  Print data as a hex string into a buffer.

- int wpa_snprintf_hex_uppercase (char ∗buf, size_t buf_size, const u8 ∗data, size_t len)

  Print data as a upper case hex string into buf.

- const char ∗ wpa_ssid_txt (u8 ∗ssid, size_t ssid_len)

  Convert SSID to a printable string.

Variables

- void typedef void(∗ **wpa_msg_cb_func**)(void ∗ctx, int level, const char ∗txt, size_t len)

6.14.1 Detailed Description

wpa_supplicant/hostapd / common helper functions, etc.

Copyright

Copyright (c) 2002-2006, Jouni Malinen <jkmaline@cc.hut.fi>

Definition in file common.h.

6.14.2 Define Documentation

6.14.2.1 #define WPA_GET_BE24(a)

Value:

```
((((u32) (a)[0]) << 16) | (((u32) (a)[1]) << 8) | \
                         ((u32) (a)[2]))
```

Definition at line 146 of file common.h.

6.14.2.2 #define WPA_GET_BE32(a)

Value:

```
((((u32) (a)[0]) << 24) | (((u32) (a)[1]) << 16) | \
                         (((u32) (a)[2]) << 8) | ((u32) (a)[3]))
```

Definition at line 155 of file common.h.

6.14.2.3 #define WPA_PUT_BE16(a, val)

Value:

```
do {                                    \
                (a)[0] = ((u16) (val)) >> 8;    \
                (a)[1] = ((u16) (val)) & 0xff;  \
        } while (0)
```

Definition at line 133 of file common.h.

6.14.2.4 #define WPA_PUT_BE24(a, val)

Value:

```
do {                                            \
                (a)[0] = (u8) (((u32) (val)) >> 16);    \
                (a)[1] = (u8) (((u32) (val)) >> 8);     \
                (a)[2] = (u8) (((u32) (val)) & 0xff);   \
        } while (0)
```

Definition at line 148 of file common.h.

6.14.2.5 #define WPA_PUT_BE32(a, val)

Value:

```
do {                                            \
                (a)[0] = (u8) (((u32) (val)) >> 24);    \
                (a)[1] = (u8) (((u32) (val)) >> 16);    \
                (a)[2] = (u8) (((u32) (val)) >> 8);     \
                (a)[3] = (u8) (((u32) (val)) & 0xff);   \
        } while (0)
```

Definition at line 157 of file common.h.

6.14.2.6 #define WPA_PUT_BE64(a, val)

Value:

```
do {                                            \
                (a)[0] = (u8) (((u64) (val)) >> 56);    \
                (a)[1] = (u8) (((u64) (val)) >> 48);    \
                (a)[2] = (u8) (((u64) (val)) >> 40);    \
                (a)[3] = (u8) (((u64) (val)) >> 32);    \
                (a)[4] = (u8) (((u64) (val)) >> 24);    \
                (a)[5] = (u8) (((u64) (val)) >> 16);    \
                (a)[6] = (u8) (((u64) (val)) >> 8);     \
                (a)[7] = (u8) (((u64) (val)) & 0xff);   \
        } while (0)
```

Definition at line 165 of file common.h.

6.14.2.7 #define WPA_PUT_LE16(a, val)

Value:

```
do {                                    \
                (a)[1] = ((u16) (val)) >> 8;    \
                (a)[0] = ((u16) (val)) & 0xff;  \
        } while (0)
```

Definition at line 140 of file common.h.

6.14.3 Function Documentation

6.14.3.1 int hexstr2bin (const char ∗ *hex*, u8 ∗ *buf*, size_t *len*)

Convert ASCII hex string into binary data.

Parameters:

hex ASCII hex string (e.g., "01ab")

buf Buffer for the binary data

len Length of the text to convert in bytes (of buf); hex will be double this size

Returns:

0 on success, -1 on failure (invalid hex string)

Definition at line 93 of file common.c.

6.14.3.2 int hwaddr_aton (const char ∗ *txt*, u8 ∗ *addr*)

Convert ASCII string to MAC address.

Parameters:

txt MAC address as a string (e.g., "00:11:22:33:44:55")

addr Buffer for the MAC address (ETH_ALEN = 6 bytes)

Returns:

0 on success, -1 on failure (e.g., string not a MAC address)

Definition at line 62 of file common.c.

6.14.3.3 void inc_byte_array (u8 ∗ *counter*, size_t *len*)

Increment arbitrary length byte array by one.

Parameters:

counter Pointer to byte array

len Length of the counter in bytes

This function increments the last byte of the counter by one and continues rolling over to more significant bytes if the byte was incremented from 0xff to 0x00.

Definition at line 121 of file common.c.

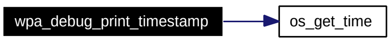
6.14.3.4 void wpa_debug_print_timestamp (void)

Print timestamp for debug output.

This function prints a timestamp in <seconds from 1970>.<microsoconds> format if debug output has been configured to include timestamps in debug messages.

Definition at line 152 of file common.c.

Here is the call graph for this function:



6.14.3.5 void void wpa_hexdump (int *level*, const char ∗ *title*, const u8 ∗ *buf*, size_t *len*)

conditional hex dump

Parameters:

level priority level (MSG_∗) of the message

title title of for the message

buf data buffer to be dumped

len length of the buf

This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration. The contents of buf is printed out has hex dump.

Definition at line 242 of file common.c.

6.14.3.6 void wpa_hexdump_ascii (int *level*, const char ∗ *title*, const u8 ∗ *buf*, size_t *len*)

conditional hex dump

Parameters:

level priority level (MSG_∗) of the message

title title of for the message

buf data buffer to be dumped

len length of the buf

This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration. The contents of buf is printed out has hex dump with both the hex numbers and ASCII characters (for printable range) are shown. 16 bytes per line will be shown.

Definition at line 339 of file common.c.

6.14.3.7 void wpa_hexdump_ascii_key (int *level*, const char ∗ *title*, const u8 ∗ *buf*, size_t *len*)

conditional hex dump, hide keys

Parameters:

level priority level (MSG_∗) of the message

title title of for the message

buf data buffer to be dumped

len length of the buf

This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration. The contents of buf is printed out has hex dump with both the hex numbers and ASCII characters (for printable range) are shown. 16 bytes per line will be shown. This works like wpa_hexdump_ascii(), but by default, does not include secret keys (passwords, etc.) in debug output.

Definition at line 345 of file common.c.

6.14.3.8 void wpa_hexdump_key (int *level*, const char ∗ *title*, const u8 ∗ *buf*, size_t *len*)

conditional hex dump, hide keys

Parameters:

level priority level (MSG_∗) of the message

title title of for the message

buf data buffer to be dumped

len length of the buf

This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration. The contents of buf is printed out has hex dump. This works like wpa_hexdump(), but by default, does not include secret keys (passwords, etc.) in debug output.

Definition at line 248 of file common.c.

6.14.3.9 void wpa_msg (void ∗ *ctx*, int *level*, char ∗ *fmt*, ...)

Conditional printf for default target and ctrl_iface monitors.

Parameters:

ctx Pointer to context data; this is the ctx variable registered with struct wpa_driver_ops::init()

level priority level (MSG_∗) of the message

fmt printf format string, followed by optional arguments

This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration. This function is like wpa_printf(), but it also sends the same message to all attached ctrl_iface monitors.

Note: New line '

' is added to the end of the text when printing to stdout.

6.14.3.10 void wpa_msg_register_cb (wpa_msg_cb_func *func*)

Register callback function for wpa_msg() messages.

Parameters:

func Callback function (NULL to unregister)

Definition at line 390 of file common.c.

6.14.3.11 void wpa_printf (int *level*, char ∗ *fmt*, ...)

conditional printf

Parameters:

level priority level (MSG_∗) of the message

fmt printf format string, followed by optional arguments

This function is used to print conditional debugging and error messages. The output may be directed to stdout, stderr, and/or syslog based on configuration.

Note: New line '

' is added to the end of the text when printing to stdout.

6.14.3.12 int wpa_snprintf_hex (char ∗ *buf*, size_t *buf_size*, const u8 ∗ *data*, size_t *len*)

Print data as a hex string into a buffer.

Parameters:

buf Memory area to use as the output buffer

buf_size Maximum buffer size in bytes (should be at least 2 ∗ len + 1)

data Data to be printed

len Length of data in bytes

Returns:

Number of bytes written

Definition at line 450 of file common.c.

6.14.3.13 int wpa_snprintf_hex_uppercase (char ∗ *buf*, size_t *buf_size*, const u8 ∗ *data*, size_t *len*)

Print data as a upper case hex string into buf.

Parameters:

buf Memory area to use as the output buffer

buf_size Maximum buffer size in bytes (should be at least 2 ∗ len + 1)

data Data to be printed

len Length of data in bytes

Returns:

Number of bytes written

Definition at line 465 of file common.c.

6.14.3.14 const char∗ wpa_ssid_txt (u8 ∗ *ssid*, size_t *ssid_len*)

Convert SSID to a printable string.

Parameters:

ssid SSID (32-octet string)

ssid_len Length of ssid in octets

Returns:

Pointer to a printable string

This function can be used to convert SSIDs into printable form. In most cases, SSIDs do not use unprintable characters, but IEEE 802.11 standard does not limit the used character set, so anything could be used in an SSID.

This function uses a static buffer, so only one call can be used at the time, i.e., this is not re-entrant and the returned buffer must be used before calling this again.

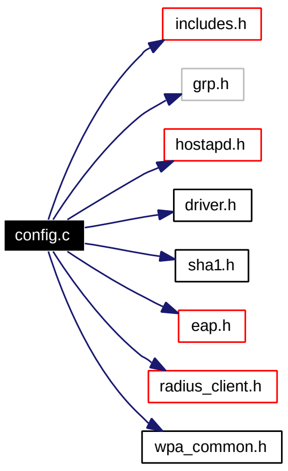
Definition at line 598 of file common.c.

6.15 config.c File Reference

hostapd / Configuration file

```
#include "includes.h"
#include <grp.h>
#include "hostapd.h"
#include "driver.h"
#include "sha1.h"
#include "eap.h"
#include "radius_client.h"
#include "wpa_common.h"
```

Include dependency graph for config.c:



Defines

- #define **MAX_STA_COUNT** 2007

Enumerations

- enum {

  IEEE80211_TX_QUEUE_DATA0 = 0, **IEEE80211_TX_QUEUE_DATA1** = 1, **IEEE80211_-TX_QUEUE_DATA2** = 2, **IEEE80211_TX_QUEUE_DATA3** = 3,

  IEEE80211_TX_QUEUE_DATA4 = 4, **IEEE80211_TX_QUEUE_AFTER_BEACON** = 6, **IEEE80211_TX_QUEUE_BEACON** = 7 }

Functions

- int **hostapd_mac_comp** (const void ∗a, const void ∗b)
- int **hostapd_mac_comp_empty** (const void ∗a)

- int **hostapd_setup_wpa_psk** (struct hostapd_bss_config ∗conf)
- hostapd_config ∗ **hostapd_config_read** (const char ∗fname)
- int **hostapd_wep_key_cmp** (struct hostapd_wep_keys ∗a, struct hostapd_wep_keys ∗b)
- void **hostapd_config_free** (struct hostapd_config ∗conf)
- int **hostapd_maclist_found** (macaddr ∗list, int num_entries, const u8 ∗addr)
- int **hostapd_rate_found** (int ∗list, int rate)
- const char ∗ **hostapd_get_vlan_id_ifname** (struct hostapd_vlan ∗vlan, int vlan_id)
- const u8 ∗ **hostapd_get_psk** (const struct hostapd_bss_config ∗conf, const u8 ∗addr, const u8 ∗prev_psk)
- const struct hostapd_eap_user ∗ **hostapd_get_eap_user** (const struct hostapd_bss_config ∗conf, const u8 ∗identity, size_t identity_len, int phase2)

6.15.1 Detailed Description

hostapd / Configuration file

Copyright

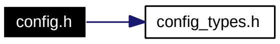
Definition in file config.c.

6.16 config.h File Reference

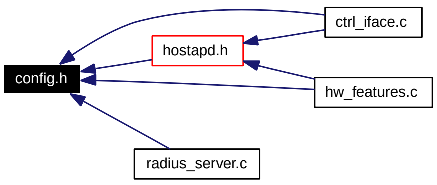
hostapd / Configuration file

```
#include "config_types.h"
```

Include dependency graph for config.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **HOSTAPD_MAX_SSID_LEN** 32
- #define **NUM_WEP_KEYS** 4
- #define **DYNAMIC_VLAN_DISABLED** 0
- #define **DYNAMIC_VLAN_OPTIONAL** 1
- #define **DYNAMIC_VLAN_REQUIRED** 2
- #define **VLAN_ID_WILDCARD** -1
- #define **PMK_LEN** 32
- #define **EAP_USER_MAX_METHODS** 8
- #define **NUM_TX_QUEUES** 8
- #define **HOSTAPD_MODULE_IEEE80211** BIT(0)
- #define **HOSTAPD_MODULE_IEEE8021X** BIT(1)
- #define **HOSTAPD_MODULE_RADIUS** BIT(2)
- #define **HOSTAPD_MODULE_WPA** BIT(3)
- #define **HOSTAPD_MODULE_DRIVER** BIT(4)
- #define **HOSTAPD_MODULE_IAPP** BIT(5)
- #define **HOSTAPD_MODULE_MLME** BIT(6)
- #define **HOSTAPD_AUTH_OPEN** BIT(0)
- #define **HOSTAPD_AUTH_SHARED_KEY** BIT(1)
- #define **HOSTAPD_WPA_VERSION_WPA** BIT(0)
- #define **HOSTAPD_WPA_VERSION_WPA2** BIT(1)
- #define **WPA_KEY_MGMT_IEEE8021X** BIT(0)
- #define **WPA_KEY_MGMT_PSK** BIT(1)
- #define **WPA_CIPHER_NONE** BIT(0)
- #define **WPA_CIPHER_WEP40** BIT(1)
- #define **WPA_CIPHER_WEP104** BIT(2)
- #define **WPA_CIPHER_TKIP** BIT(3)
- #define **WPA_CIPHER_CCMP** BIT(4)

Typedefs

- typedef u8 **macaddr** [ETH_ALEN]
- typedef enum hostap_security_policy **secpolicy**

Enumerations

- enum **hostap_security_policy** {

  SECURITY_PLAINTEXT = 0, **SECURITY_STATIC_WEP** = 1, **SECURITY_IEEE_802_1X** = 2, **SECURITY_WPA_PSK** = 3,

  SECURITY_WPA = 4 }
- enum **hostapd_hw_mode** { **HOSTAPD_MODE_IEEE80211B**, **HOSTAPD_MODE_-IEEE80211G**, **HOSTAPD_MODE_IEEE80211A**, **NUM_HOSTAPD_MODES** }

Functions

- int **hostapd_mac_comp** (const void ∗a, const void ∗b)
- int **hostapd_mac_comp_empty** (const void ∗a)
- hostapd_config ∗ **hostapd_config_read** (const char ∗fname)
- void **hostapd_config_free** (struct hostapd_config ∗conf)
- int **hostapd_maclist_found** (macaddr ∗list, int num_entries, const u8 ∗addr)
- int **hostapd_rate_found** (int ∗list, int rate)
- int **hostapd_wep_key_cmp** (struct hostapd_wep_keys ∗a, struct hostapd_wep_keys ∗b)
- const u8 ∗ **hostapd_get_psk** (const struct hostapd_bss_config ∗conf, const u8 ∗addr, const u8 ∗prev_psk)
- int **hostapd_setup_wpa_psk** (struct hostapd_bss_config ∗conf)
- const char ∗ **hostapd_get_vlan_id_ifname** (struct hostapd_vlan ∗vlan, int vlan_id)
- const struct hostapd_eap_user ∗ **hostapd_get_eap_user** (const struct hostapd_bss_config ∗conf, const u8 ∗identity, size_t identity_len, int phase2)

6.16.1 Detailed Description

hostapd / Configuration file

Copyright

Copyright (c) 2003-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

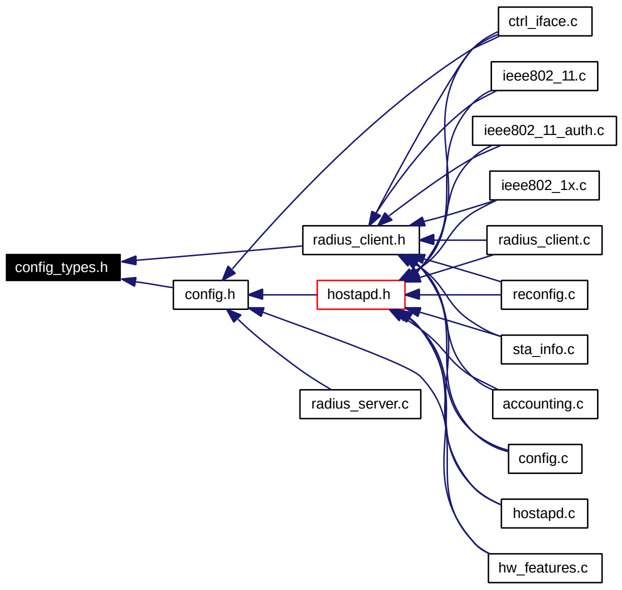
See README and COPYING for more details.

Definition in file config.h.

6.17 config_types.h File Reference

hostapd / Shared configuration file defines

This graph shows which files directly or indirectly include this file:



6.17.1 Detailed Description

hostapd / Shared configuration file defines

Copyright

Copyright (c) 2003-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file config_types.h.

6.18 crypto.c File Reference

WPA Supplicant / wrapper functions for libcrypto.

`#include "includes.h"`

`#include <openssl/opensslv.h>`

`#include <openssl/md4.h>`

`#include <openssl/md5.h>`

`#include <openssl/sha.h>`

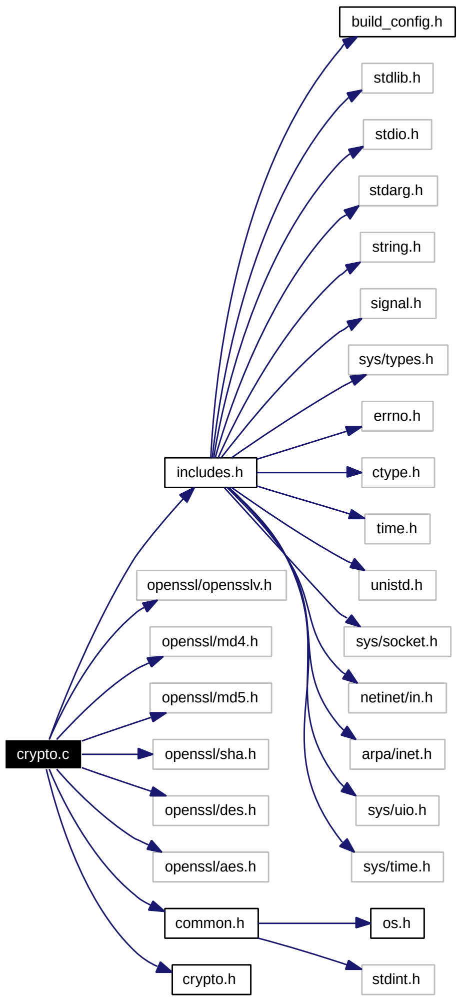
`#include <openssl/des.h>`

`#include <openssl/aes.h>`

`#include "common.h"`

`#include "crypto.h"`

Include dependency graph for crypto.c:



Defines

- #define **DES_key_schedule** des_key_schedule
- #define **DES_cblock** des_cblock
- #define **DES_set_key**(key, schedule) des_set_key((key), ∗(schedule))
- #define **DES_ecb_encrypt**(input, output, ks, enc) des_ecb_encrypt((input), (output), ∗(ks), (enc))

Functions

- void md4_vector (size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

  MD4 hash for data vector.

- void des_encrypt (const u8 ∗clear, const u8 ∗key, u8 ∗cypher)

  Encrypt one block with DES.

6.18.1 Detailed Description

WPA Supplicant / wrapper functions for libcrypto.

Copyright

Copyright (c) 2004-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file crypto.c.

6.18.2 Function Documentation

6.18.2.1 void des_encrypt (const u8 ∗ *clear*, const u8 ∗ *key*, u8 ∗ *cypher*)

Encrypt one block with DES.

Parameters:

clear 8 octets (in)

key 7 octets (in) (no parity bits included)

cypher 8 octets (out)

Definition at line 48 of file crypto.c.

6.18.2.2 void md4_vector (size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

MD4 hash for data vector.

Parameters:

num_elem Number of elements in the data vector

addr Pointers to the data areas

len Lengths of the data blocks

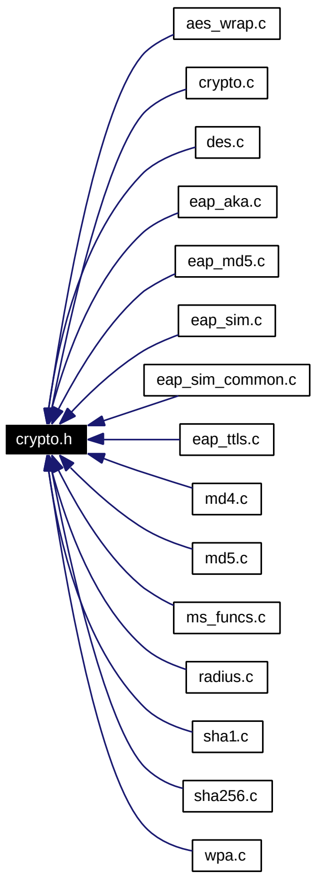
mac Buffer for the hash

Definition at line 36 of file crypto.c.

6.19 crypto.h File Reference

WPA Supplicant / wrapper functions for crypto libraries.

This graph shows which files directly or indirectly include this file:



Enumerations

- enum **crypto_hash_alg** { **CRYPTO_HASH_ALG_MD5**, **CRYPTO_HASH_ALG_SHA1**, **CRYPTO_HASH_ALG_HMAC_MD5**, **CRYPTO_HASH_ALG_HMAC_SHA1** }
- enum **crypto_cipher_alg** {

  CRYPTO_CIPHER_NULL = 0, **CRYPTO_CIPHER_ALG_AES**, **CRYPTO_CIPHER_ALG_-3DES**, **CRYPTO_CIPHER_ALG_DES**,

  CRYPTO_CIPHER_ALG_RC2, **CRYPTO_CIPHER_ALG_RC4** }

Functions

- void md4_vector (size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

  MD4 hash for data vector.

- void md5_vector (size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

  MD5 hash for data vector.

- void sha1_vector (size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

  SHA-1 hash for data vector.

- int fips186_2_prf (const u8 ∗seed, size_t seed_len, u8 ∗x, size_t xlen)

  NIST FIPS Publication 186-2 change notice 1 PRF.

- void sha256_vector (size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

  SHA256 hash for data vector.

- void des_encrypt (const u8 ∗clear, const u8 ∗key, u8 ∗cypher)

  Encrypt one block with DES.

- void ∗ aes_encrypt_init (const u8 ∗key, size_t len)

  Initialize AES for encryption.

- void aes_encrypt (void ∗ctx, const u8 ∗plain, u8 ∗crypt)

  Encrypt one AES block.

- void aes_encrypt_deinit (void ∗ctx)

  Deinitialize AES encryption.

- void ∗ aes_decrypt_init (const u8 ∗key, size_t len)

  Initialize AES for decryption.

- void aes_decrypt (void ∗ctx, const u8 ∗crypt, u8 ∗plain)

  Decrypt one AES block.

- void aes_decrypt_deinit (void ∗ctx)

  Deinitialize AES decryption.

- crypto_hash ∗ crypto_hash_init (enum crypto_hash_alg alg, const u8 ∗key, size_t key_len)

  Initialize hash/HMAC function.

- void crypto_hash_update (struct crypto_hash ∗ctx, const u8 ∗data, size_t len)

  Add data to hash calculation.

- int crypto_hash_finish (struct crypto_hash ∗ctx, u8 ∗hash, size_t ∗len)

  Complete hash calculation.

- crypto_cipher ∗ crypto_cipher_init (enum crypto_cipher_alg alg, const u8 ∗iv, const u8 ∗key, size_t key_len)

  Initialize block/stream cipher function.

- int crypto_cipher_encrypt (struct crypto_cipher ∗ctx, const u8 ∗plain, u8 ∗crypt, size_t len)

  Cipher encrypt.

- int crypto_cipher_decrypt (struct crypto_cipher ∗ctx, const u8 ∗crypt, u8 ∗plain, size_t len)

  Cipher decrypt.

- void crypto_cipher_deinit (struct crypto_cipher ∗ctx)

  Free cipher context.

- crypto_public_key ∗ crypto_public_key_import (const u8 ∗key, size_t len)

  Import an RSA public key.

- crypto_private_key ∗ crypto_private_key_import (const u8 ∗key, size_t len)

  Import an RSA private key.

- crypto_public_key ∗ crypto_public_key_from_cert (const u8 ∗buf, size_t len)

  Import an RSA public key from a certificate.

- int crypto_public_key_encrypt_pkcs1_v15 (struct crypto_public_key ∗key, const u8 ∗in, size_t inlen, u8 ∗out, size_t ∗outlen)

  Public key encryption (PKCS #1 v1.5).

- int crypto_private_key_sign_pkcs1 (struct crypto_private_key ∗key, const u8 ∗in, size_t inlen, u8 ∗out, size_t ∗outlen)

  Sign with private key (PKCS #1).

- void crypto_public_key_free (struct crypto_public_key ∗key)

  Free public key.

- void crypto_private_key_free (struct crypto_private_key ∗key)

  Free private key.

- int crypto_public_key_decrypt_pkcs1 (struct crypto_public_key ∗key, const u8 ∗crypt, size_t crypt_-len, u8 ∗plain, size_t ∗plain_len)

  Decrypt PKCS #1 signature.

- int crypto_global_init (void)

  Initialize crypto wrapper.

- void crypto_global_deinit (void)

  Deinitialize crypto wrapper.

- int crypto_mod_exp (const u8 ∗base, size_t base_len, const u8 ∗power, size_t power_len, const u8 ∗modulus, size_t modulus_len, u8 ∗result, size_t ∗result_len)

  Modular exponentiation of large integers.

6.19.1 Detailed Description

WPA Supplicant / wrapper functions for crypto libraries.

Copyright

This file defines the cryptographic functions that need to be implemented for wpa_supplicant and hostapd. When TLS is not used, internal implementation of MD5, SHA1, and AES is used and no external libraries are required. When TLS is enabled (e.g., by enabling EAP-TLS or EAP-PEAP), the crypto library used by the TLS implementation is expected to be used for non-TLS needs, too, in order to save space by not implementing these functions twice.

Wrapper code for using each crypto library is in its own file (crypto∗.c) and one of these files is build and linked in to provide the functions defined here.

Definition in file crypto.h.

6.19.2 Function Documentation

6.19.2.1 void aes_decrypt (void ∗ *ctx*, const u8 ∗ *crypt*, u8 ∗ *plain*)

Decrypt one AES block.

Parameters:

ctx Context pointer from aes_encrypt_init()

crypt Encrypted data (16 bytes)

plain Buffer for the decrypted data (16 bytes)

Definition at line 1099 of file aes.c.

6.19.2.2 void aes_decrypt_deinit (void ∗ *ctx*)

Deinitialize AES decryption.

Parameters:

ctx Context pointer from aes_encrypt_init()

Definition at line 1105 of file aes.c.

6.19.2.3 void∗ aes_decrypt_init (const u8 ∗ *key*, size_t *len*)

Initialize AES for decryption.

Parameters:

key Decryption key

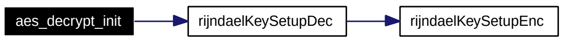
len Key length in bytes (usually 16, i.e., 128 bits)

Returns:

Pointer to context data or NULL on failure

Definition at line 1086 of file aes.c.

Here is the call graph for this function:



6.19.2.4 void aes_encrypt (void ∗ *ctx*, const u8 ∗ *plain*, u8 ∗ *crypt*)

Encrypt one AES block.

Parameters:

ctx Context pointer from aes_encrypt_init()

plain Plaintext data to be encrypted (16 bytes)

crypt Buffer for the encrypted data (16 bytes)

Definition at line 1074 of file aes.c.

6.19.2.5 void aes_encrypt_deinit (void ∗ *ctx*)

Deinitialize AES encryption.

Parameters:

ctx Context pointer from aes_encrypt_init()

Definition at line 1080 of file aes.c.

6.19.2.6 void∗ aes_encrypt_init (const u8 ∗ *key*, size_t *len*)

Initialize AES for encryption.

Parameters:

key Encryption key

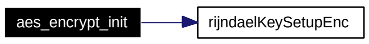
len Key length in bytes (usually 16, i.e., 128 bits)

Returns:

Pointer to context data or NULL on failure

Definition at line 1061 of file aes.c.

Here is the call graph for this function:



6.19.2.7 int crypto_cipher_decrypt (struct crypto_cipher ∗ *ctx*, const u8 ∗ *crypt*, u8 ∗ *plain*, size_t *len*)

Cipher decrypt.

Parameters:

ctx Context pointer from crypto_cipher_init()

crypt Ciphertext to decrypt

plain Resulting plaintext

len Length of the cipher text

Returns:

0 on success, -1 on failure

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.8 void crypto_cipher_deinit (struct crypto_cipher ∗ *ctx*)

Free cipher context.

Parameters:

ctx Context pointer from crypto_cipher_init()

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.9 int crypto_cipher_encrypt (struct crypto_cipher ∗ *ctx*, const u8 ∗ *plain*, u8 ∗ *crypt*, size_t *len*)

Cipher encrypt.

Parameters:

ctx Context pointer from crypto_cipher_init()

plain Plaintext to cipher

crypt Resulting ciphertext

len Length of the plaintext

Returns:

0 on success, -1 on failure

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.10 struct crypto_cipher∗ crypto_cipher_init (enum crypto_cipher_alg *alg*, const u8 ∗ *iv*, const u8 ∗ *key*, size_t *key_len*)

Initialize block/stream cipher function.

Parameters:

alg Cipher algorithm

iv Initialization vector for block ciphers or NULL for stream ciphers

key Cipher key

key_len Length of key in bytes

Returns:

Pointer to cipher context to use with other cipher functions or NULL on failure

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.11 void crypto_global_deinit (void)

Deinitialize crypto wrapper.

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.12 int crypto_global_init (void)

Initialize crypto wrapper.

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.13 int crypto_hash_finish (struct crypto_hash ∗ *ctx*, u8 ∗ *hash*, size_t ∗ *len*)

Complete hash calculation.

Parameters:

ctx Context pointer from crypto_hash_init()

hash Buffer for hash value or NULL if caller is just freeing the hash context

len Pointer to length of the buffer or NULL if caller is just freeing the hash context; on return, this is set to the actual length of the hash value

Returns:

0 on success, -1 if buffer is too small (len set to needed length), or -2 on other failures (including failed crypto_hash_update() operations)

This function calculates the hash value and frees the context buffer that was used for hash calculation.

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.14 struct crypto_hash∗ crypto_hash_init (enum crypto_hash_alg *alg*, const u8 ∗ *key*, size_t *key_len*)

Initialize hash/HMAC function.

Parameters:

alg Hash algorithm

key Key for keyed hash (e.g., HMAC) or NULL if not needed

key_len Length of the key in bytes

Returns:

Pointer to hash context to use with other hash functions or NULL on failure

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.15 void crypto_hash_update (struct crypto_hash ∗ *ctx*, const u8 ∗ *data*, size_t *len*)

Add data to hash calculation.

Parameters:

ctx Context pointer from crypto_hash_init()

data Data buffer to add

len Length of the buffer

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.16 int crypto_mod_exp (const u8 ∗ *base*, size_t *base_len*, const u8 ∗ *power*, size_t *power_len*, const u8 ∗ *modulus*, size_t *modulus_len*, u8 ∗ *result*, size_t ∗ *result_len*)

Modular exponentiation of large integers.

Parameters:

base Base integer (big endian byte array)

base_len Length of base integer in bytes

power Power integer (big endian byte array)

power_len Length of power integer in bytes

modulus Modulus integer (big endian byte array)

modulus_len Length of modulus integer in bytes

result Buffer for the result

result_len Result length (max buffer size on input, real len on output)

Returns:

0 on success, -1 on failure

This function calculates result = base ^ power mod modulus. modules_len is used as the maximum size of modulus buffer. It is set to the used size on success.

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.17 void crypto_private_key_free (struct crypto_private_key ∗ *key*)

Free private key.

Parameters:

key Private key from crypto_private_key_import()

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.18 struct crypto_private_key∗ crypto_private_key_import (const u8 ∗ *key*, size_t *len*)

Import an RSA private key.

Parameters:

key Key buffer (DER encoded RSA private key)

len Key buffer length in bytes

Returns:

Pointer to the private key or NULL on failure

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.19 int crypto_private_key_sign_pkcs1 (struct crypto_private_key ∗ *key*, const u8 ∗ *in*, size_t *inlen*, u8 ∗ *out*, size_t ∗ *outlen*)

Sign with private key (PKCS #1).

Parameters:

key Private key from crypto_private_key_import()

in Plaintext buffer

inlen Length of plaintext buffer in bytes

out Output buffer for encrypted (signed) data

outlen Length of output buffer in bytes; set to used length on success

Returns:

0 on success, -1 on failure

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.20 int crypto_public_key_decrypt_pkcs1 (struct crypto_public_key ∗ *key*, const u8 ∗ *crypt*, size_t *crypt_len*, u8 ∗ *plain*, size_t ∗ *plain_len*)

Decrypt PKCS #1 signature.

Parameters:

key Public key

crypt Encrypted signature data (using the private key)

crypt_len Encrypted signature data length

plain Buffer for plaintext (at least crypt_len bytes)

plain_len Plaintext length (max buffer size on input, real len on output);

Returns:
0 on success, -1 on failure

6.19.2.21 int crypto_public_key_encrypt_pkcs1_v15 (struct crypto_public_key ∗ *key*, const u8 ∗ *in*, size_t *inlen*, u8 ∗ *out*, size_t ∗ *outlen*)

Public key encryption (PKCS #1 v1.5).

Parameters:

key Public key

in Plaintext buffer

inlen Length of plaintext buffer in bytes

out Output buffer for encrypted data

outlen Length of output buffer in bytes; set to used length on success

Returns:
0 on success, -1 on failure

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.22 void crypto_public_key_free (struct crypto_public_key ∗ *key*)

Free public key.

Parameters:

key Public key

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.23 struct crypto_public_key∗ crypto_public_key_from_cert (const u8 ∗ *buf*, size_t *len*)

Import an RSA public key from a certificate.

Parameters:

buf DER encoded X.509 certificate

len Certificate buffer length in bytes

Returns:
Pointer to public key or NULL on failure

This function can just return NULL if the crypto library does not support X.509 parsing. In that case, internal code will be used to parse the certificate and public key is imported using crypto_public_key_-import().

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.24 struct crypto_public_key∗ crypto_public_key_import (const u8 ∗ *key*, size_t *len*)

Import an RSA public key.

Parameters:

key Key buffer (DER encoded RSA public key)

len Key buffer length in bytes

Returns:

Pointer to the public key or NULL on failure

This function can just return NULL if the crypto library supports X.509 parsing. In that case, crypto_-public_key_from_cert() is used to import the public key from a certificate.

This function is only used with internal TLSv1 implementation (CONFIG_TLS=internal). If that is not used, the crypto wrapper does not need to implement this.

6.19.2.25 void des_encrypt (const u8 ∗ *clear*, const u8 ∗ *key*, u8 ∗ *cypher*)

Encrypt one block with DES.

Parameters:

clear 8 octets (in)

key 7 octets (in) (no parity bits included)

cypher 8 octets (out)

Definition at line 48 of file crypto.c.

6.19.2.26 int fips186_2_prf (const u8 ∗ *seed*, size_t *seed_len*, u8 ∗ *x*, size_t *xlen*)

NIST FIPS Publication 186-2 change notice 1 PRF.

Parameters:

seed Seed/key for the PRF

seed_len Seed length in bytes

x Buffer for PRF output

xlen Output length in bytes

Returns:

0 on success, -1 on failure

This function implements random number generation specified in NIST FIPS Publication 186-2 for EAP-SIM. This PRF uses a function that is similar to SHA-1, but has different message padding.

6.19.2.27 void md4_vector (size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

MD4 hash for data vector.

Parameters:

num_elem Number of elements in the data vector

addr Pointers to the data areas

len Lengths of the data blocks

mac Buffer for the hash

Definition at line 36 of file crypto.c.

6.19.2.28 void md5_vector (size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

MD5 hash for data vector.

Parameters:

num_elem Number of elements in the data vector

addr Pointers to the data areas

len Lengths of the data blocks

mac Buffer for the hash

6.19.2.29 void sha1_vector (size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

SHA-1 hash for data vector.

Parameters:

num_elem Number of elements in the data vector

addr Pointers to the data areas

len Lengths of the data blocks

mac Buffer for the hash

6.19.2.30 void sha256_vector (size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

SHA256 hash for data vector.

Parameters:

num_elem Number of elements in the data vector

addr Pointers to the data areas

len Lengths of the data blocks

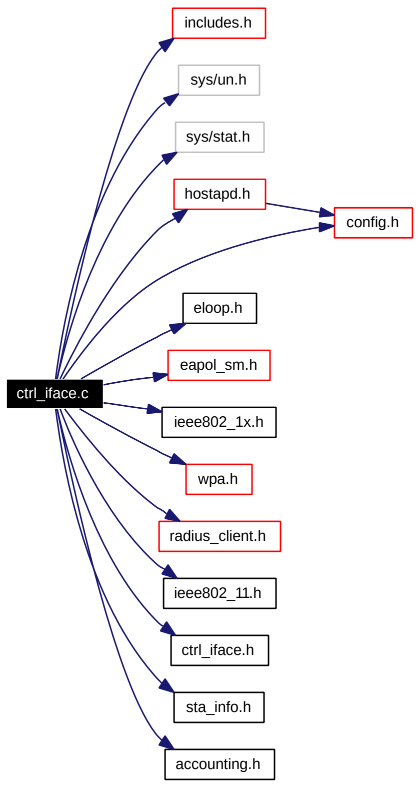
mac Buffer for the hash

6.20 ctrl_iface.c File Reference

hostapd / UNIX domain socket -based control interface

```
#include "includes.h"
#include <sys/un.h>
#include <sys/stat.h>
#include "hostapd.h"
#include "eloop.h"
#include "config.h"
#include "eapol_sm.h"
#include "ieee802_1x.h"
#include "wpa.h"
#include "radius_client.h"
#include "ieee802_11.h"
#include "ctrl_iface.h"
#include "sta_info.h"
#include "accounting.h"
```

Include dependency graph for ctrl_iface.c:



Functions

- int **hostapd_ctrl_iface_init** (struct hostapd_data ∗hapd)
- void **hostapd_ctrl_iface_deinit** (struct hostapd_data ∗hapd)
- void **hostapd_ctrl_iface_send** (struct hostapd_data ∗hapd, int level, char ∗buf, size_t len)

6.20.1 Detailed Description

hostapd / UNIX domain socket -based control interface

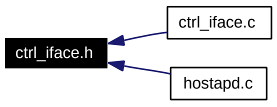
Copyright

Definition in file ctrl_iface.c.

6.21 ctrl_iface.h File Reference

hostapd / UNIX domain socket -based control interface

This graph shows which files directly or indirectly include this file:



Functions

- int **hostapd_ctrl_iface_init** (struct hostapd_data ∗hapd)
- void **hostapd_ctrl_iface_deinit** (struct hostapd_data ∗hapd)
- void **hostapd_ctrl_iface_send** (struct hostapd_data ∗hapd, int level, char ∗buf, size_t len)

6.21.1 Detailed Description

hostapd / UNIX domain socket -based control interface

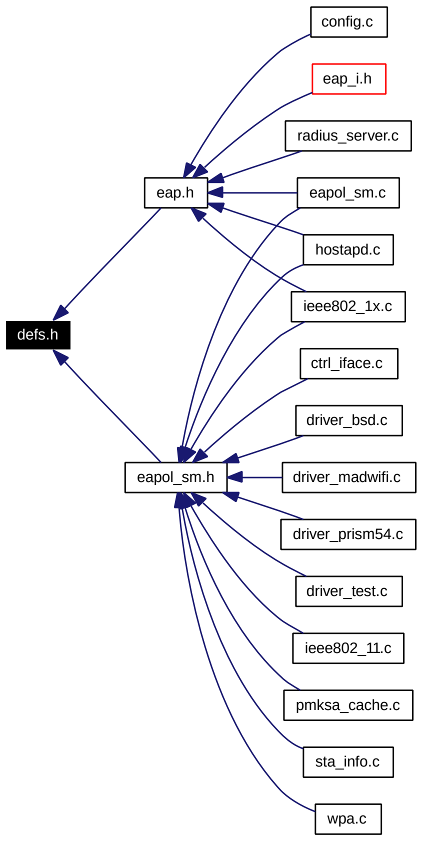
Copyright

Definition in file ctrl_iface.h.

6.22 defs.h File Reference

WPA Supplicant - Common definitions.

This graph shows which files directly or indirectly include this file:



Defines

- #define **MLME_SETPROTECTION_PROTECT_TYPE_NONE** 0
- #define **MLME_SETPROTECTION_PROTECT_TYPE_RX** 1
- #define **MLME_SETPROTECTION_PROTECT_TYPE_TX** 2
- #define **MLME_SETPROTECTION_PROTECT_TYPE_RX_TX** 3
- #define **MLME_SETPROTECTION_KEY_TYPE_GROUP** 0
- #define **MLME_SETPROTECTION_KEY_TYPE_PAIRWISE** 1

Enumerations

- enum **Boolean** { **FALSE** = 0, **TRUE** = 1 }
- enum **wpa_alg** {

  WPA_ALG_NONE, **WPA_ALG_WEP**, **WPA_ALG_TKIP**, **WPA_ALG_CCMP**,

  WPA_ALG_IGTK, **WPA_ALG_DHV** }

- enum **wpa_cipher** {

  CIPHER_NONE, **CIPHER_WEP40**, **CIPHER_TKIP**, **CIPHER_CCMP**,

  CIPHER_WEP104 }
- enum **wpa_key_mgmt** {

  KEY_MGMT_802_1X, **KEY_MGMT_PSK**, **KEY_MGMT_NONE**, **KEY_MGMT_802_1X_-NO_WPA**,

  KEY_MGMT_WPA_NONE }
- enum wpa_states {

  WPA_DISCONNECTED, WPA_INACTIVE, WPA_SCANNING, WPA_ASSOCIATING,

  WPA_ASSOCIATED, WPA_4WAY_HANDSHAKE, WPA_GROUP_HANDSHAKE, WPA_-COMPLETED }

6.22.1 Detailed Description

WPA Supplicant - Common definitions.

Copyright

Copyright (c) 2004-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file defs.h.

6.22.2 Enumeration Type Documentation

6.22.2.1 enum wpa_states

enum wpa_states - wpa_supplicant state

These enumeration values are used to indicate the current wpa_supplicant state (wpa_s->wpa_state). The current state can be retrieved with wpa_supplicant_get_state() function and the state can be changed by calling wpa_supplicant_set_state(). In WPA state machine (wpa.c and preauth.c), the wrapper functions wpa_sm_get_state() and wpa_sm_set_state() should be used to access the state variable.

Enumeration values:

WPA_DISCONNECTED Disconnected state.

This state indicates that client is not associated, but is likely to start looking for an access point. This state is entered when a connection is lost.

WPA_INACTIVE Inactive state (wpa_supplicant disabled).

This state is entered if there are no enabled networks in the configuration. wpa_supplicant is not trying to associate with a new network and external interaction (e.g., ctrl_iface call to add or enable a network) is needed to start association.

WPA_SCANNING Scanning for a network.

This state is entered when wpa_supplicant starts scanning for a network.

WPA_ASSOCIATING Trying to associate with a BSS/SSID.

This state is entered when wpa_supplicant has found a suitable BSS to associate with and the driver is configured to try to associate with this BSS in ap_scan=1 mode. When using ap_scan=2 mode, this state is entered when the driver is configured to try to associate with a network using the configured SSID and security policy.

WPA_ASSOCIATED Association completed.

This state is entered when the driver reports that association has been successfully completed with an AP. If IEEE 802.1X is used (with or without WPA/WPA2), wpa_supplicant remains in this state until the IEEE 802.1X/EAPOL authentication has been completed.

WPA_4WAY_HANDSHAKE WPA 4-Way Key Handshake in progress.

This state is entered when WPA/WPA2 4-Way Handshake is started. In case of WPA-PSK, this happens when receiving the first EAPOL-Key frame after association. In case of WPA-EAP, this state is entered when the IEEE 802.1X/EAPOL authentication has been completed.

WPA_GROUP_HANDSHAKE WPA Group Key Handshake in progress.

This state is entered when 4-Way Key Handshake has been completed (i.e., when the supplicant sends out message 4/4) and when Group Key rekeying is started by the AP (i.e., when supplicant receives message 1/2).

WPA_COMPLETED All authentication completed.

This state is entered when the full authentication process is completed. In case of WPA2, this happens when the 4-Way Handshake is successfully completed. With WPA, this state is entered after the Group Key Handshake; with IEEE 802.1X (non-WPA) connection is completed after dynamic keys are received (or if not used, after the EAP authentication has been completed). With static WEP keys and plaintext connections, this state is entered when an association has been completed.

This state indicates that the supplicant has completed its processing for the association phase and that data connection is fully configured.

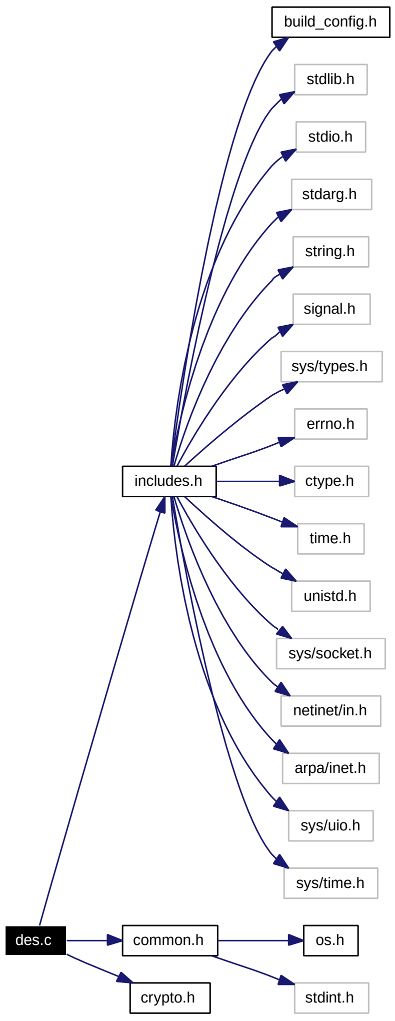
Definition at line 45 of file defs.h.

6.23 des.c File Reference

DES and 3DES-EDE ciphers.

```
#include "includes.h"
#include "common.h"
#include "crypto.h"
```

Include dependency graph for des.c:



6.23.1 Detailed Description

DES and 3DES-EDE ciphers.

Modifications to LibTomCrypt implementation:

Copyright

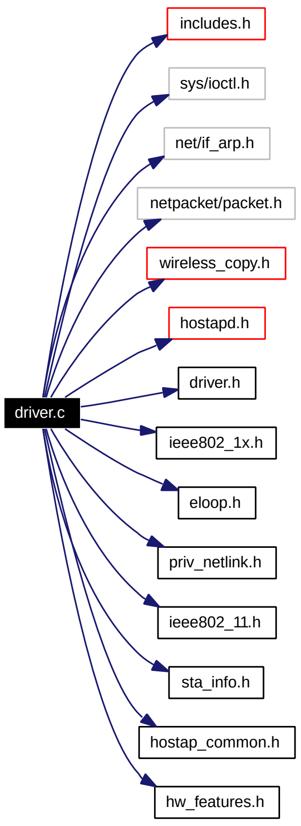
Definition in file des.c.

6.24 driver.c File Reference

hostapd / Kernel driver communication with Linux Host AP driver

```
#include "includes.h"
#include <sys/ioctl.h>
#include <net/if_arp.h>
#include <netpacket/packet.h>
#include "wireless_copy.h"
#include "hostapd.h"
#include "driver.h"
#include "ieee802_1x.h"
#include "eloop.h"
#include "priv_netlink.h"
#include "ieee802_11.h"
#include "sta_info.h"
#include "hostap_common.h"
#include "hw_features.h"
```

Include dependency graph for driver.c:



Defines

- #define **WLAN_RATE_1M** BIT(0)
- #define **WLAN_RATE_2M** BIT(1)
- #define **WLAN_RATE_5M5** BIT(2)
- #define **WLAN_RATE_11M** BIT(3)

Functions

- void **hostap_driver_register** (void)

6.24.1 Detailed Description

hostapd / Kernel driver communication with Linux Host AP driver

Copyright

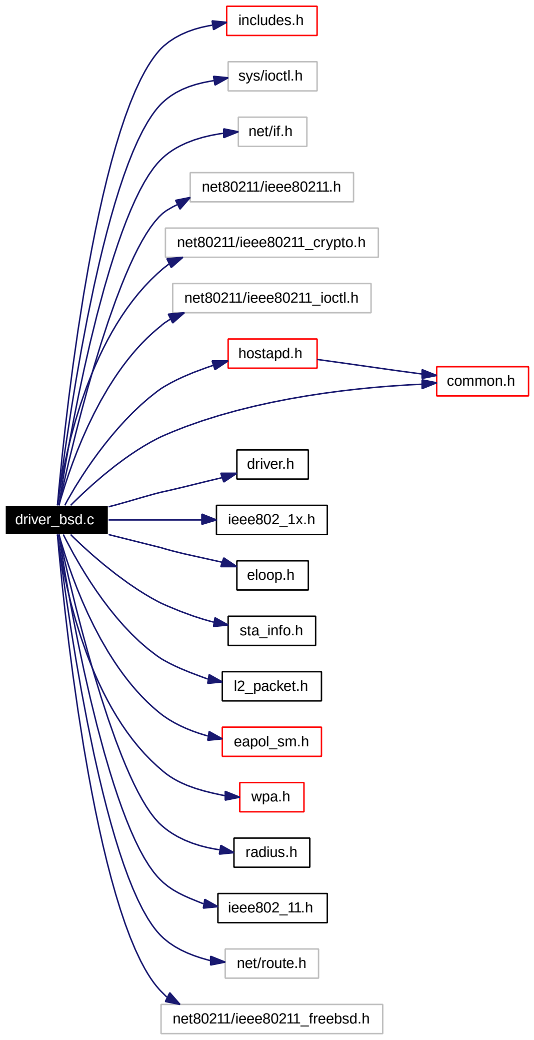
Definition in file driver.c.

6.25 driver_bsd.c File Reference

hostapd / Driver interaction with BSD net80211 layer

```
#include "includes.h"
#include <sys/ioctl.h>
#include <net/if.h>
#include <net80211/ieee80211.h>
#include <net80211/ieee80211_crypto.h>
#include <net80211/ieee80211_ioctl.h>
#include "hostapd.h"
#include "driver.h"
#include "ieee802_1x.h"
#include "eloop.h"
#include "sta_info.h"
#include "l2_packet.h"
#include "eapol_sm.h"
#include "wpa.h"
#include "radius.h"
#include "ieee802_11.h"
#include "common.h"
#include <net/route.h>
#include <net80211/ieee80211_freebsd.h>
```

Include dependency graph for driver_bsd.c:



Functions

- void **bsd_driver_register** (void)

6.25.1 Detailed Description

hostapd / Driver interaction with BSD net80211 layer

Copyright

Copyright (c) 2004, Sam Leffler <`sam@errno.com`> Copyright (c) 2004, 2Wire, Inc

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

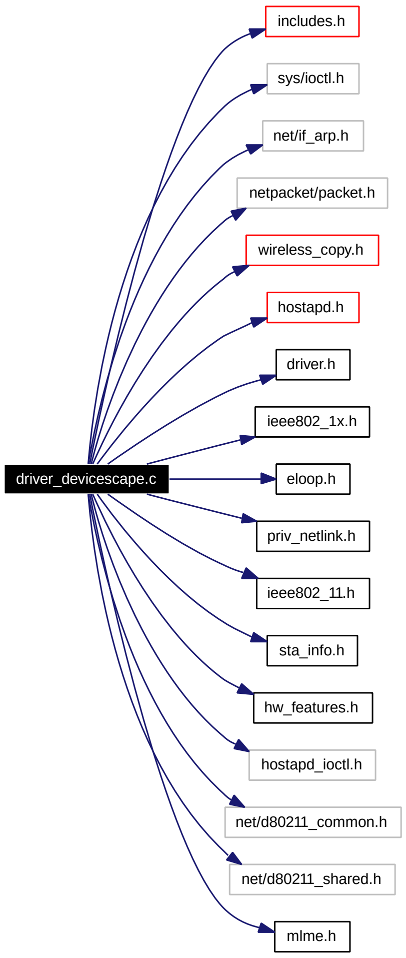
Definition in file driver_bsd.c.

6.26 driver_devicescape.c File Reference

hostapd / Kernel driver communication with Devicescape IEEE 802.11 stack

```
#include "includes.h"
#include <sys/ioctl.h>
#include <net/if_arp.h>
#include <netpacket/packet.h>
#include "wireless_copy.h"
#include "hostapd.h"
#include "driver.h"
#include "ieee802_1x.h"
#include "eloop.h"
#include "priv_netlink.h"
#include "ieee802_11.h"
#include "sta_info.h"
#include "hw_features.h"
#include <hostapd_ioctl.h>
#include <net/d80211_common.h>
#include <net/d80211_shared.h>
#include "mlme.h"
```

Include dependency graph for driver_devicescape.c:



Defines

- #define **HAPD_DECL** struct hostapd_data ∗hapd = iface → bss[0]

Functions

- void **devicescape_driver_register** (void)

6.26.1 Detailed Description

hostapd / Kernel driver communication with Devicescape IEEE 802.11 stack

Copyright

See README and COPYING for more details.

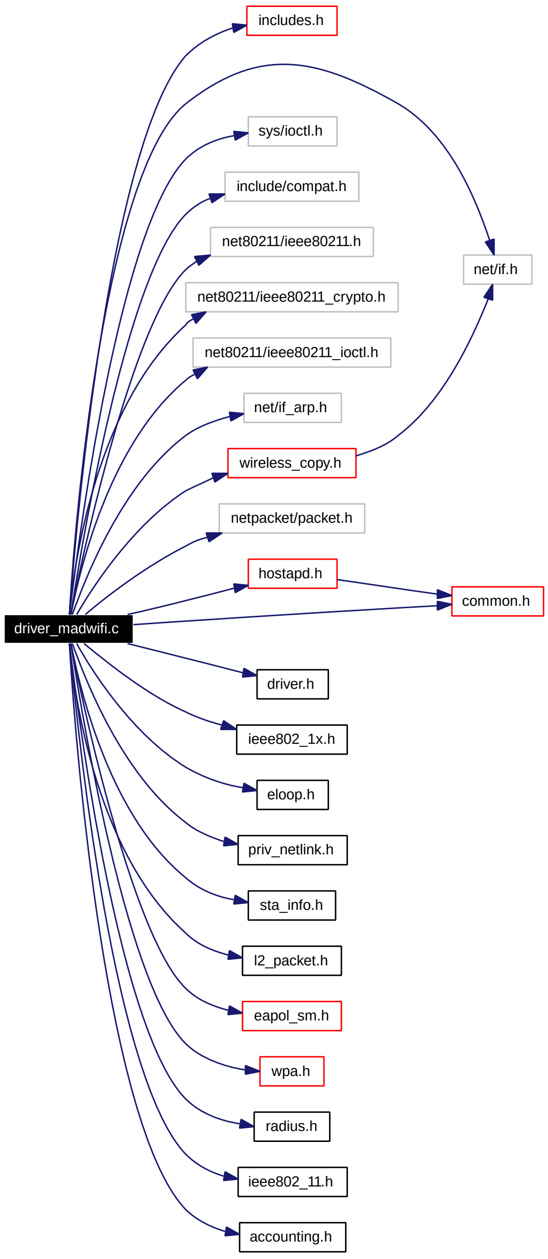
Definition in file driver_devicescape.c.

6.27 driver_madwifi.c File Reference

hostapd / Driver interaction with MADWIFI 802.11 driver

```
#include "includes.h"
#include <net/if.h>
#include <sys/ioctl.h>
#include <include/compat.h>
#include <net80211/ieee80211.h>
#include <net80211/ieee80211_crypto.h>
#include <net80211/ieee80211_ioctl.h>
#include <net/if_arp.h>
#include "wireless_copy.h"
#include <netpacket/packet.h>
#include "hostapd.h"
#include "driver.h"
#include "ieee802_1x.h"
#include "eloop.h"
#include "priv_netlink.h"
#include "sta_info.h"
#include "l2_packet.h"
#include "eapol_sm.h"
#include "wpa.h"
#include "radius.h"
#include "ieee802_11.h"
#include "accounting.h"
#include "common.h"
```

Include dependency graph for driver_madwifi.c:



Functions

- void **madwifi_driver_register** (void)

6.27.1 Detailed Description

hostapd / Driver interaction with MADWIFI 802.11 driver

Copyright

Copyright (c) 2004, Sam Leffler <`sam@errno.com`> Copyright (c) 2004, Video54 Technologies Copyright (c) 2005-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

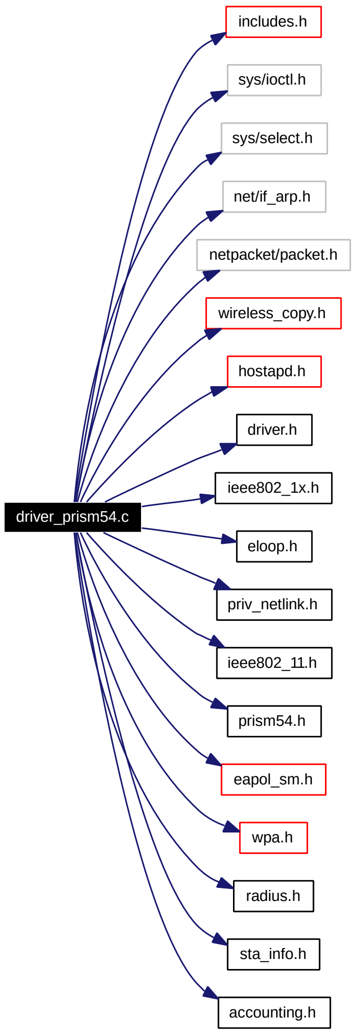
Definition in file driver_madwifi.c.

6.28 driver_prism54.c File Reference

hostapd / Driver interaction with Prism54 PIMFOR interface

```
#include "includes.h"
#include <sys/ioctl.h>
#include <sys/select.h>
#include <net/if_arp.h>
#include <netpacket/packet.h>
#include "wireless_copy.h"
#include "hostapd.h"
#include "driver.h"
#include "ieee802_1x.h"
#include "eloop.h"
#include "priv_netlink.h"
#include "ieee802_11.h"
#include "prism54.h"
#include "eapol_sm.h"
#include "wpa.h"
#include "radius.h"
#include "sta_info.h"
#include "accounting.h"
```

Include dependency graph for driver_prism54.c:



Functions

- void **prism54_driver_register** (void)

Variables

- const int **PIM_BUF_SIZE** = 4096

6.28.1 Detailed Description

hostapd / Driver interaction with Prism54 PIMFOR interface

Copyright

Copyright (c) 2004, Bell Kin <`bell_kin@pek.com.tw`> based on hostap driver.c, ieee802_11.c

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

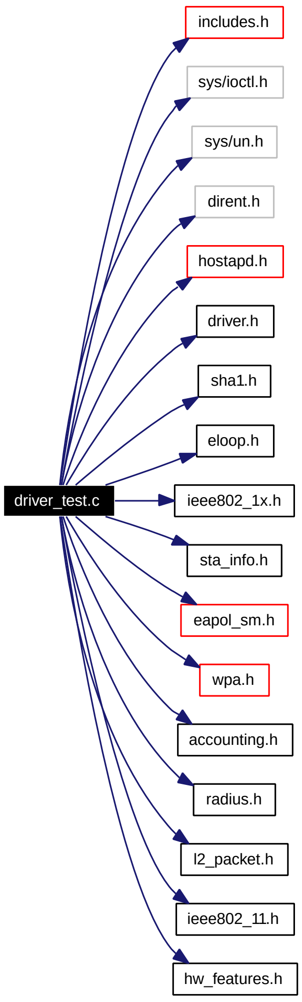
Definition in file driver_prism54.c.

6.29 driver_test.c File Reference

hostapd / Driver interface for development testing

```
#include "includes.h"
#include <sys/ioctl.h>
#include <sys/un.h>
#include <dirent.h>
#include "hostapd.h"
#include "driver.h"
#include "sha1.h"
#include "eloop.h"
#include "ieee802_1x.h"
#include "sta_info.h"
#include "eapol_sm.h"
#include "wpa.h"
#include "accounting.h"
#include "radius.h"
#include "l2_packet.h"
#include "ieee802_11.h"
#include "hw_features.h"
```

Include dependency graph for driver_test.c:



Functions

- void **test_driver_register** (void)

6.29.1 Detailed Description

hostapd / Driver interface for development testing

Copyright

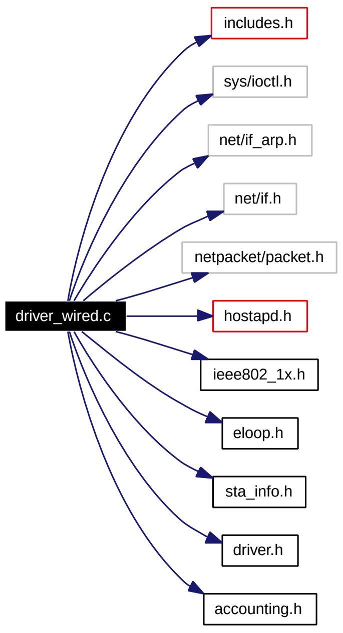
Definition in file driver_test.c.

6.30 driver_wired.c File Reference

hostapd / Kernel driver communication for wired (Ethernet) drivers

```
#include "includes.h"
#include <sys/ioctl.h>
#include <net/if_arp.h>
#include <net/if.h>
#include <netpacket/packet.h>
#include "hostapd.h"
#include "ieee802_1x.h"
#include "eloop.h"
#include "sta_info.h"
#include "driver.h"
#include "accounting.h"
```

Include dependency graph for driver_wired.c:



Defines

- #define **WIRED_EAPOL_MULTICAST_GROUP** {0x01,0x80,0xc2,0x00,0x00,0x03}

Functions

- void **wired_driver_register** (void)

6.30.1 Detailed Description

hostapd / Kernel driver communication for wired (Ethernet) drivers

Copyright

Copyright (c) 2002-2005, Jouni Malinen <jkmaline@cc.hut.fi> Copyright (c) 2004, Gunter Burchardt <tira@isx.de>

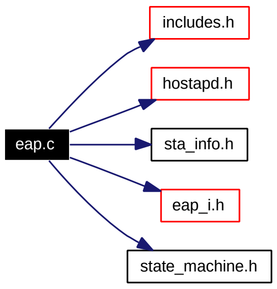
Definition in file driver_wired.c.

6.31 eap.c File Reference

hostapd / EAP Standalone Authenticator state machine (RFC 4137)

```
#include "includes.h"
#include "hostapd.h"
#include "sta_info.h"
#include "eap_i.h"
#include "state_machine.h"
```

Include dependency graph for eap.c:



Defines

- #define **STATE_MACHINE_DATA** struct eap_sm
- #define **STATE_MACHINE_DEBUG_PREFIX** "EAP"
- #define **EAP_MAX_AUTH_ROUNDS** 50

Functions

- int eap_user_get (struct eap_sm ∗sm, const u8 ∗identity, size_t identity_len, int phase2)

  Fetch user information from the database.

- **SM_STATE** (EAP, DISABLED)
- **SM_STATE** (EAP, INITIALIZE)
- **SM_STATE** (EAP, PICK_UP_METHOD)
- **SM_STATE** (EAP, IDLE)
- **SM_STATE** (EAP, RETRANSMIT)
- **SM_STATE** (EAP, RECEIVED)
- **SM_STATE** (EAP, DISCARD)
- **SM_STATE** (EAP, SEND_REQUEST)
- **SM_STATE** (EAP, INTEGRITY_CHECK)
- **SM_STATE** (EAP, METHOD_REQUEST)
- **SM_STATE** (EAP, METHOD_RESPONSE)
- **SM_STATE** (EAP, PROPOSE_METHOD)
- **SM_STATE** (EAP, NAK)
- **SM_STATE** (EAP, SELECT_ACTION)
- **SM_STATE** (EAP, TIMEOUT_FAILURE)

- **SM_STATE** (EAP, FAILURE)
- **SM_STATE** (EAP, SUCCESS)
- **SM_STEP** (EAP)
- void eap_sm_process_nak (struct eap_sm ∗sm, u8 ∗nak_list, size_t len)

  Process EAP-Response/Nak.

- int eap_sm_step (struct eap_sm ∗sm)

  Step EAP state machine.

- void eap_set_eapRespData (struct eap_sm ∗sm, const u8 ∗eapRespData, size_t eapRespDataLen)

  Set EAP response (eapRespData).

- eap_sm ∗ eap_sm_init (void ∗eapol_ctx, struct eapol_callbacks ∗eapol_cb, struct eap_config ∗conf)

  Allocate and initialize EAP state machine.

- void eap_sm_deinit (struct eap_sm ∗sm)

  Deinitialize and free an EAP state machine.

- void eap_sm_notify_cached (struct eap_sm ∗sm)

  Notify EAP state machine of cached PMK.

- void eap_sm_pending_cb (struct eap_sm ∗sm)

  EAP state machine callback for a pending EAP request.

- int eap_sm_method_pending (struct eap_sm ∗sm)

  Query whether EAP method is waiting for pending data.

- const u8 ∗ eap_hdr_validate (int vendor, EapType eap_type, const u8 ∗msg, size_t msglen, size_t ∗plen)

  Validate EAP header.

- eap_hdr ∗ eap_msg_alloc (int vendor, EapType type, size_t ∗len, size_t payload_len, u8 code, u8 identifier, u8 ∗∗payload)

  Allocate a buffer for an EAP message.

6.31.1 Detailed Description

hostapd / EAP Standalone Authenticator state machine (RFC 4137)

Copyright

Copyright (c) 2004-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eap.c.

6.31.2 Function Documentation

6.31.2.1 const u8∗ eap_hdr_validate (int *vendor*, EapType *eap_type*, const u8 ∗ *msg*, size_t *msglen*, size_t ∗ *plen*)

Validate EAP header.

Parameters:

vendor Expected EAP Vendor-Id (0 = IETF)

eap_type Expected EAP type number

msg EAP frame (starting with EAP header)

msglen Length of msg

plen Pointer to variable to contain the returned payload length

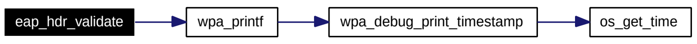
Returns:

Pointer to EAP payload (after type field), or NULL on failure

This is a helper function for EAP method implementations. This is usually called in the beginning of struct eap_method::process() function to verify that the received EAP request packet has a valid header. This function is able to process both legacy and expanded EAP headers and in most cases, the caller can just use the returned payload pointer (into ∗plen) for processing the payload regardless of whether the packet used the expanded EAP header or not.

Definition at line 1048 of file eap.c.

Here is the call graph for this function:



6.31.2.2 struct eap_hdr∗ eap_msg_alloc (int *vendor*, EapType *type*, size_t ∗ *len*, size_t *payload_len*, u8 *code*, u8 *identifier*, u8 ∗∗ *payload*)

Allocate a buffer for an EAP message.

Parameters:

vendor Vendor-Id (0 = IETF)

type EAP type

len Buffer for returning message length

payload_len Payload length in bytes (data after Type)

code Message Code (EAP_CODE_∗)

identifier Identifier

payload Pointer to payload pointer that will be set to point to the beginning of the payload or NULL if payload pointer is not needed

Returns:

Pointer to the allocated message buffer or NULL on error

This function can be used to allocate a buffer for an EAP message and fill in the EAP header. This function is automatically using expanded EAP header if the selected Vendor-Id is not IETF. In other words, most EAP methods do not need to separately select which header type to use when using this function to allocate the message buffers.

Definition at line 1121 of file eap.c.

6.31.2.3 void eap_set_eapRespData (struct eap_sm ∗ *sm*, const u8 ∗ *eapRespData*, size_t *eapRespDataLen*)

Set EAP response (eapRespData).

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

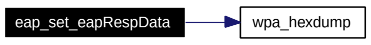
eapRespData EAP-Response payload from the supplicant

eapRespDataLen Length of eapRespData in bytes

This function is called when an EAP-Response is received from a supplicant.

Definition at line 900 of file eap.c.

Here is the call graph for this function:



6.31.2.4 void eap_sm_deinit (struct eap_sm ∗ *sm*)

Deinitialize and free an EAP state machine.

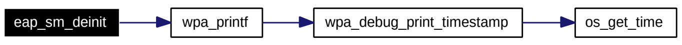
Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

This function deinitializes EAP state machine and frees all allocated resources.

Definition at line 965 of file eap.c.

Here is the call graph for this function:



6.31.2.5 struct eap_sm∗ eap_sm_init (void ∗ *eapol_ctx*, struct eapol_callbacks ∗ *eapol_cb*, struct eap_config ∗ *conf*)

Allocate and initialize EAP state machine.

Parameters:

eapol_ctx Context data to be used with eapol_cb calls

eapol_cb Pointer to EAPOL callback functions

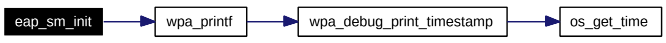
conf EAP configuration

Returns:
Pointer to the allocated EAP state machine or NULL on failure

This function allocates and initializes an EAP state machine.

Definition at line 936 of file eap.c.

Here is the call graph for this function:



6.31.2.6 int eap_sm_method_pending (struct eap_sm ∗ *sm*)

Query whether EAP method is waiting for pending data.

Parameters:
sm Pointer to EAP state machine allocated with eap_sm_init()

Returns:
1 if method is waiting for pending data or 0 if not

Definition at line 1022 of file eap.c.

6.31.2.7 void eap_sm_notify_cached (struct eap_sm ∗ *sm*)

Notify EAP state machine of cached PMK.

Parameters:
sm Pointer to EAP state machine allocated with eap_sm_init()

This function is called when PMKSA caching is used to skip EAP authentication.

Definition at line 990 of file eap.c.

6.31.2.8 void eap_sm_pending_cb (struct eap_sm ∗ *sm*)

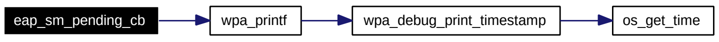
EAP state machine callback for a pending EAP request.

Parameters:
sm Pointer to EAP state machine allocated with eap_sm_init()

This function is called when data for a pending EAP-Request is received.

Definition at line 1006 of file eap.c.

Here is the call graph for this function:



6.31.2.9 void eap_sm_process_nak (struct eap_sm * *sm*, u8 * *nak_list*, size_t *len*)

Process EAP-Response/Nak.

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

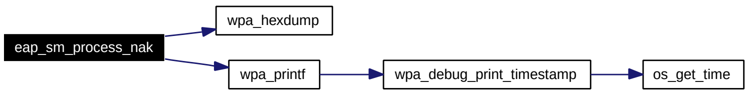
nak_list Nak list (allowed methods) from the supplicant

len Length of nak_list in bytes

This function is called when EAP-Response/Nak is received from the supplicant. This can happen for both phase 1 and phase 2 authentications.

Definition at line 714 of file eap.c.

Here is the call graph for this function:



6.31.2.10 int eap_sm_step (struct eap_sm * *sm*)

Step EAP state machine.

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

Returns:

1 if EAP state was changed or 0 if not

This function advances EAP state machine to a new state to match with the current variables. This should be called whenever variables used by the EAP state machine have changed.

Definition at line 878 of file eap.c.

6.31.2.11 int eap_user_get (struct eap_sm * *sm*, const u8 * *identity*, size_t *identity_len*, int *phase2*)

Fetch user information from the database.

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

identity Identity (User-Name) of the user

identity_len Length of identity in bytes

phase2 0 = EAP phase1 user, 1 = EAP phase2 (tunneled) user

Returns:

0 on success, or -1 on failure

This function is used to fetch user information for EAP. The user will be selected based on the specified identity. sm->user and sm->user_eap_method_index are updated for the new user when a matching user is found. sm->user can be used to get user information (e.g., password).

Definition at line 91 of file eap.c.

6.32 eap.h File Reference

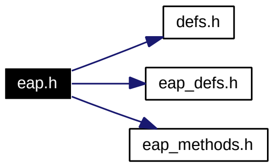
hostapd / EAP Standalone Authenticator state machine (RFC 4137)

`#include "defs.h"`

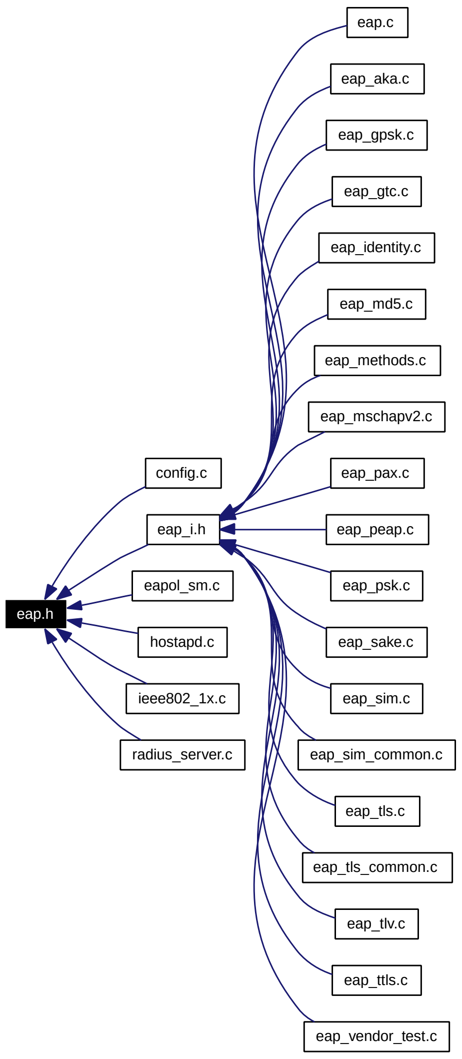
`#include "eap_defs.h"`

`#include "eap_methods.h"`

Include dependency graph for eap.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **EAP_MAX_METHODS** 8

Enumerations

- enum **eapol_bool_var** {
  EAPOL_eapSuccess, **EAPOL_eapRestart**, **EAPOL_eapFail**, **EAPOL_eapResp**,
  EAPOL_eapReq, **EAPOL_eapNoReq**, **EAPOL_portEnabled**, **EAPOL_eapTimeout** }

Functions

- eap_sm ∗ eap_sm_init (void ∗eapol_ctx, struct eapol_callbacks ∗eapol_cb, struct eap_config ∗eap_-conf)

  Allocate and initialize EAP state machine.
- void eap_sm_deinit (struct eap_sm ∗sm)

  Deinitialize and free an EAP state machine.
- int eap_sm_step (struct eap_sm ∗sm)

  Step EAP state machine.
- void eap_set_eapRespData (struct eap_sm ∗sm, const u8 ∗eapRespData, size_t eapRespDataLen)

  Set EAP response (eapRespData).
- void eap_sm_notify_cached (struct eap_sm ∗sm)

  Notify EAP state machine of cached PMK.
- void eap_sm_pending_cb (struct eap_sm ∗sm)

  EAP state machine callback for a pending EAP request.
- int eap_sm_method_pending (struct eap_sm ∗sm)

  Query whether EAP method is waiting for pending data.

6.32.1 Detailed Description

hostapd / EAP Standalone Authenticator state machine (RFC 4137)

Copyright

Definition in file eap.h.

6.32.2 Function Documentation

6.32.2.1 void eap_set_eapRespData (struct eap_sm ∗ *sm*, const u8 ∗ *eapRespData*, size_t *eapRespDataLen*)

Set EAP response (eapRespData).

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

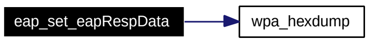
eapRespData EAP-Response payload from the supplicant

eapRespDataLen Length of eapRespData in bytes

This function is called when an EAP-Response is received from a supplicant.

Definition at line 900 of file eap.c.

Here is the call graph for this function:



6.32.2.2 void eap_sm_deinit (struct eap_sm ∗ *sm*)

Deinitialize and free an EAP state machine.

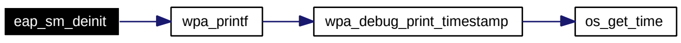
Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

This function deinitializes EAP state machine and frees all allocated resources.

Definition at line 965 of file eap.c.

Here is the call graph for this function:



6.32.2.3 struct eap_sm∗ eap_sm_init (void ∗ *eapol_ctx*, struct eapol_callbacks ∗ *eapol_cb*, struct eap_config ∗ *conf*)

Allocate and initialize EAP state machine.

Parameters:

eapol_ctx Context data to be used with eapol_cb calls

eapol_cb Pointer to EAPOL callback functions

conf EAP configuration

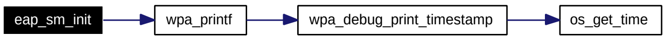
Returns:

Pointer to the allocated EAP state machine or NULL on failure

This function allocates and initializes an EAP state machine.

Definition at line 936 of file eap.c.

Here is the call graph for this function:



6.32.2.4 int eap_sm_method_pending (struct eap_sm ∗ *sm*)

Query whether EAP method is waiting for pending data.

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

Returns:

1 if method is waiting for pending data or 0 if not

Definition at line 1022 of file eap.c.

6.32.2.5 void eap_sm_notify_cached (struct eap_sm ∗ *sm*)

Notify EAP state machine of cached PMK.

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

This function is called when PMKSA caching is used to skip EAP authentication.

Definition at line 990 of file eap.c.

6.32.2.6 void eap_sm_pending_cb (struct eap_sm ∗ *sm*)

EAP state machine callback for a pending EAP request.

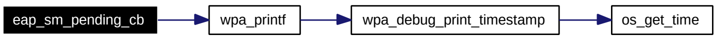
Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

This function is called when data for a pending EAP-Request is received.

Definition at line 1006 of file eap.c.

Here is the call graph for this function:



6.32.2.7 int eap_sm_step (struct eap_sm ∗ *sm*)

Step EAP state machine.

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

Returns:

1 if EAP state was changed or 0 if not

This function advances EAP state machine to a new state to match with the current variables. This should be called whenever variables used by the EAP state machine have changed.

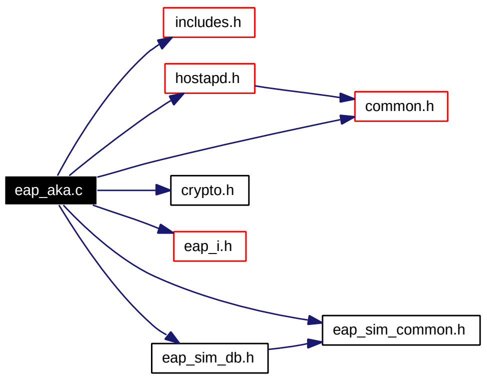
Definition at line 878 of file eap.c.

6.33 eap_aka.c File Reference

hostapd / EAP-AKA (RFC 4187)

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "crypto.h"
#include "eap_i.h"
#include "eap_sim_common.h"
#include "eap_sim_db.h"
```

Include dependency graph for eap_aka.c:



Functions

- int **eap_server_aka_register** (void)

6.33.1 Detailed Description

hostapd / EAP-AKA (RFC 4187)

Copyright

Copyright (c) 2005-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

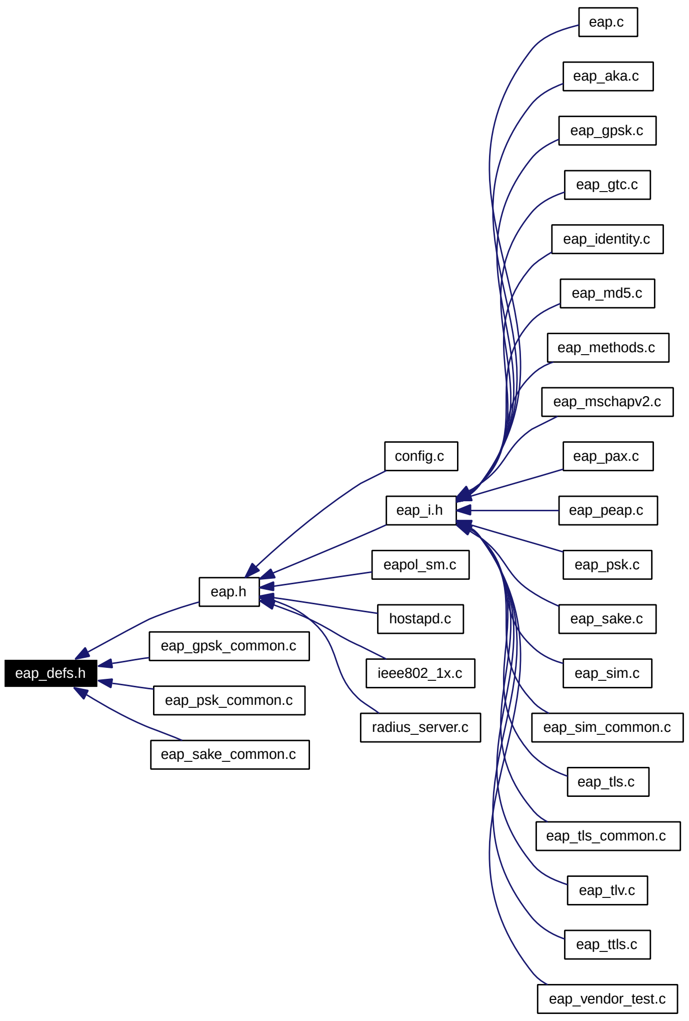
See README and COPYING for more details.

Definition in file eap_aka.c.

6.34 eap_defs.h File Reference

EAP server/peer: Shared EAP definitions.

This graph shows which files directly or indirectly include this file:



Defines

- #define **EAP_MSK_LEN** 64
- #define **EAP_EMSK_LEN** 64

Enumerations

- enum { **EAP_CODE_REQUEST** = 1, **EAP_CODE_RESPONSE** = 2, **EAP_CODE_SUCCESS** = 3, **EAP_CODE_FAILURE** = 4 }
- enum **EapType** {

  EAP_TYPE_NONE = 0, **EAP_TYPE_IDENTITY** = 1, **EAP_TYPE_NOTIFICATION** = 2, **EAP_TYPE_NAK** = 3,

EAP_TYPE_MD5 = 4, **EAP_TYPE_OTP** = 5, **EAP_TYPE_GTC** = 6, **EAP_TYPE_TLS** = 13,
EAP_TYPE_LEAP = 17, **EAP_TYPE_SIM** = 18, **EAP_TYPE_TTLS** = 21, **EAP_TYPE_AKA** = 23,
EAP_TYPE_PEAP = 25, **EAP_TYPE_MSCHAPV2** = 26, **EAP_TYPE_TLV** = 33, **EAP_-TYPE_FAST** = 43,
EAP_TYPE_PAX = 46, **EAP_TYPE_PSK** = 47, **EAP_TYPE_SAKE** = 48, **EAP_TYPE_-EXPANDED** = 254,
EAP_TYPE_GPSK = 255 }

- enum { **EAP_VENDOR_IETF** = 0 }

Variables

- eap_hdr **STRUCT_PACKED**

6.34.1 Detailed Description

EAP server/peer: Shared EAP definitions.

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

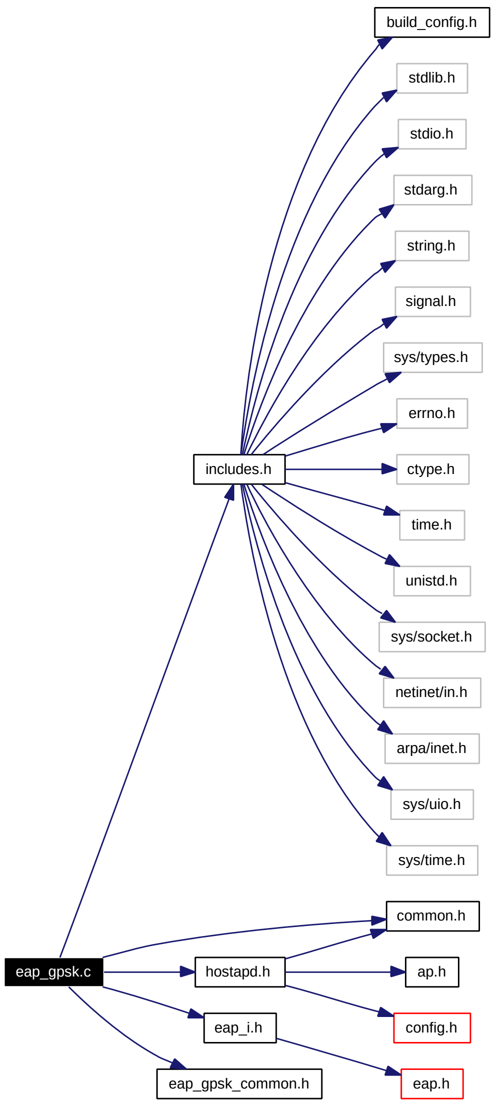
Definition in file eap_defs.h.

6.35 eap_gpsk.c File Reference

hostapd / EAP-GPSK (draft-ietf-emu-eap-gpsk-01.txt) server

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
#include "eap_gpsk_common.h"
```

Include dependency graph for eap_gpsk.c:



Defines

- #define **MAX_NUM_CSUITES** 2

Functions

- int **eap_server_gpsk_register** (void)

6.35.1 Detailed Description

hostapd / EAP-GPSK (draft-ietf-emu-eap-gpsk-01.txt) server

Copyright

Copyright (c) 2006, Jouni Malinen <jkmaline@cc.hut.fi>

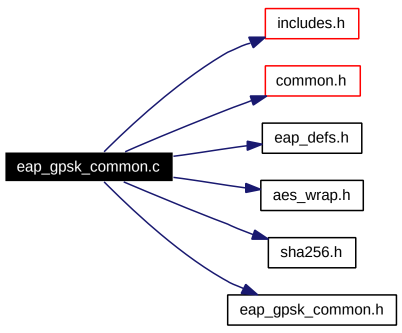
Definition in file eap_gpsk.c.

6.36 eap_gpsk_common.c File Reference

EAP server/peer: EAP-GPSK shared routines.

```
#include "includes.h"
#include "common.h"
#include "eap_defs.h"
#include "aes_wrap.h"
#include "sha256.h"
#include "eap_gpsk_common.h"
```

Include dependency graph for eap_gpsk_common.c:



Defines

- #define **EAP_GPSK_SK_LEN_AES** 16
- #define **EAP_GPSK_PK_LEN_AES** 16

Functions

- int eap_gpsk_supported_ciphersuite (int vendor, int specifier)

  Check whether ciphersuite is supported.

- int eap_gpsk_derive_keys (const u8 ∗psk, size_t psk_len, int vendor, int specifier, const u8 ∗rand_-client, const u8 ∗rand_server, const u8 ∗id_client, size_t id_client_len, const u8 ∗id_server, size_t id_server_len, u8 ∗msk, u8 ∗emsk, u8 ∗sk, size_t ∗sk_len, u8 ∗pk, size_t ∗pk_len)

  Derive EAP-GPSK keys.

- size_t eap_gpsk_mic_len (int vendor, int specifier)

  Get the length of the MIC.

- int eap_gpsk_compute_mic (const u8 ∗sk, size_t sk_len, int vendor, int specifier, const u8 ∗data, size_t len, u8 ∗mic)

  Compute EAP-GPSK MIC for an EAP packet.

6.36.1 Detailed Description

EAP server/peer: EAP-GPSK shared routines.

Copyright

Copyright (c) 2006, Jouni Malinen <jkmaline@cc.hut.fi>

Definition in file eap_gpsk_common.c.

6.36.2 Function Documentation

6.36.2.1 int eap_gpsk_compute_mic (const u8 ∗ *sk*, size_t *sk_len*, int *vendor*, int *specifier*, const u8 ∗ *data*, size_t *len*, u8 ∗ *mic*)

Compute EAP-GPSK MIC for an EAP packet.

Parameters:

sk Session key SK from eap_gpsk_derive_keys()

sk_len SK length in bytes from eap_gpsk_derive_keys()

vendor CSuite/Vendor

specifier CSuite/Specifier

data Input data to MIC

len Input data length in bytes

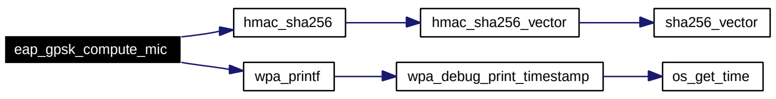
mic Buffer for the computed MIC, eap_gpsk_mic_len(cipher) bytes

Returns:

0 on success, -1 on failure

Definition at line 434 of file eap_gpsk_common.c.

Here is the call graph for this function:



6.36.2.2 int eap_gpsk_derive_keys (const u8 ∗ *psk*, size_t *psk_len*, int *vendor*, int *specifier*, const u8 ∗ *rand_client*, const u8 ∗ *rand_server*, const u8 ∗ *id_client*, size_t *id_client_len*, const u8 ∗ *id_server*, size_t *id_server_len*, u8 ∗ *msk*, u8 ∗ *emsk*, u8 ∗ *sk*, size_t ∗ *sk_len*, u8 ∗ *pk*, size_t ∗ *pk_len*)

Derive EAP-GPSK keys.

Parameters:

psk Pre-shared key (at least 16 bytes if AES is used)

psk_len Length of psk in bytes

vendor CSuite/Vendor

specifier CSuite/Specifier

rand_client 32-byte RAND_Client

rand_server 32-byte RAND_Server

id_client ID_Client

id_client_len Length of ID_Client

id_server ID_Server

id_server_len Length of ID_Server

msk Buffer for 64-byte MSK

emsk Buffer for 64-byte EMSK

sk Buffer for SK (at least EAP_GPSK_MAX_SK_LEN bytes)

sk_len Buffer for returning length of SK

pk Buffer for SK (at least EAP_GPSK_MAX_PK_LEN bytes)

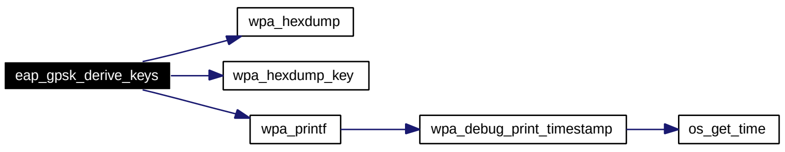
pk_len Buffer for returning length of PK

Returns:

0 on success, -1 on failure

Definition at line 318 of file eap_gpsk_common.c.

Here is the call graph for this function:



6.36.2.3 size_t eap_gpsk_mic_len (int *vendor*, int *specifier*)

Get the length of the MIC.

Parameters:

vendor CSuite/Vendor

specifier CSuite/Specifier

Returns:

MIC length in bytes

Definition at line 391 of file eap_gpsk_common.c.

6.36.2.4 int eap_gpsk_supported_ciphersuite (int *vendor*, int *specifier*)

Check whether ciphersuite is supported.

Parameters:

vendor CSuite/Vendor

specifier CSuite/Specifier

Returns:

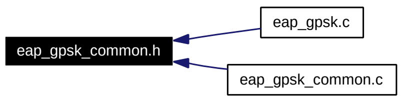
1 if ciphersuite is support, or 0 if not

Definition at line 32 of file eap_gpsk_common.c.

6.37 eap_gpsk_common.h File Reference

EAP server/peer: EAP-GPSK shared routines.

This graph shows which files directly or indirectly include this file:



Defines

- #define **EAP_GPSK_OPCODE_GPSK_1** 1
- #define **EAP_GPSK_OPCODE_GPSK_2** 2
- #define **EAP_GPSK_OPCODE_GPSK_3** 3
- #define **EAP_GPSK_OPCODE_GPSK_4** 4
- #define **EAP_GPSK_RAND_LEN** 32
- #define **EAP_GPSK_MAX_SK_LEN** 32
- #define **EAP_GPSK_MAX_PK_LEN** 32
- #define **EAP_GPSK_MAX_MIC_LEN** 32
- #define **EAP_GPSK_VENDOR_IETF** 0x000000
- #define **EAP_GPSK_CIPHER_RESERVED** 0x000000
- #define **EAP_GPSK_CIPHER_AES** 0x000001
- #define **EAP_GPSK_CIPHER_SHA256** 0x000002

Functions

- int eap_gpsk_supported_ciphersuite (int vendor, int specifier)

  Check whether ciphersuite is supported.

- int eap_gpsk_derive_keys (const u8 ∗psk, size_t psk_len, int vendor, int specifier, const u8 ∗rand_-client, const u8 ∗rand_server, const u8 ∗id_client, size_t id_client_len, const u8 ∗id_server, size_t id_server_len, u8 ∗msk, u8 ∗emsk, u8 ∗sk, size_t ∗sk_len, u8 ∗pk, size_t ∗pk_len)

  Derive EAP-GPSK keys.

- size_t eap_gpsk_mic_len (int vendor, int specifier)

  Get the length of the MIC.

- int eap_gpsk_compute_mic (const u8 ∗sk, size_t sk_len, int vendor, int specifier, const u8 ∗data, size_t len, u8 ∗mic)

  Compute EAP-GPSK MIC for an EAP packet.

Variables

- eap_gpsk_csuite **STRUCT_PACKED**

6.37.1 Detailed Description

EAP server/peer: EAP-GPSK shared routines.

Copyright

Copyright (c) 2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

Definition in file eap_gpsk_common.h.

6.37.2 Function Documentation

6.37.2.1 int eap_gpsk_compute_mic (const u8 ∗ *sk*, size_t *sk_len*, int *vendor*, int *specifier*, const u8 ∗ *data*, size_t *len*, u8 ∗ *mic*)

Compute EAP-GPSK MIC for an EAP packet.

Parameters:

sk Session key SK from eap_gpsk_derive_keys()

sk_len SK length in bytes from eap_gpsk_derive_keys()

vendor CSuite/Vendor

specifier CSuite/Specifier

data Input data to MIC

len Input data length in bytes

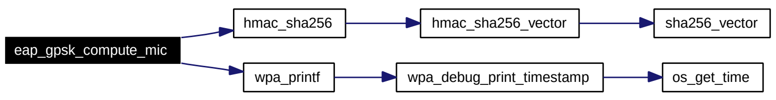
mic Buffer for the computed MIC, eap_gpsk_mic_len(cipher) bytes

Returns:

0 on success, -1 on failure

Definition at line 434 of file eap_gpsk_common.c.

Here is the call graph for this function:



6.37.2.2 int eap_gpsk_derive_keys (const u8 ∗ *psk*, size_t *psk_len*, int *vendor*, int *specifier*, const u8 ∗ *rand_client*, const u8 ∗ *rand_server*, const u8 ∗ *id_client*, size_t *id_client_len*, const u8 ∗ *id_server*, size_t *id_server_len*, u8 ∗ *msk*, u8 ∗ *emsk*, u8 ∗ *sk*, size_t ∗ *sk_len*, u8 ∗ *pk*, size_t ∗ *pk_len*)

Derive EAP-GPSK keys.

Parameters:

psk Pre-shared key (at least 16 bytes if AES is used)

psk_len Length of psk in bytes

vendor CSuite/Vendor

specifier CSuite/Specifier

rand_client 32-byte RAND_Client

rand_server 32-byte RAND_Server

id_client ID_Client

id_client_len Length of ID_Client

id_server ID_Server

id_server_len Length of ID_Server

msk Buffer for 64-byte MSK

emsk Buffer for 64-byte EMSK

sk Buffer for SK (at least EAP_GPSK_MAX_SK_LEN bytes)

sk_len Buffer for returning length of SK

pk Buffer for SK (at least EAP_GPSK_MAX_PK_LEN bytes)

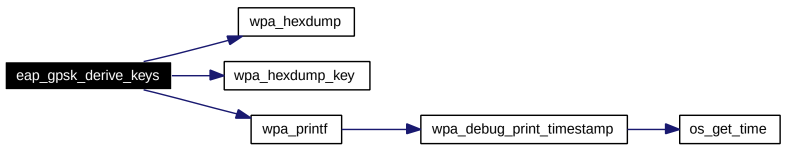
pk_len Buffer for returning length of PK

Returns:

0 on success, -1 on failure

Definition at line 318 of file eap_gpsk_common.c.

Here is the call graph for this function:



6.37.2.3 size_t eap_gpsk_mic_len (int *vendor*, int *specifier*)

Get the length of the MIC.

Parameters:

vendor CSuite/Vendor

specifier CSuite/Specifier

Returns:

MIC length in bytes

Definition at line 391 of file eap_gpsk_common.c.

6.37.2.4 int eap_gpsk_supported_ciphersuite (int *vendor*, int *specifier*)

Check whether ciphersuite is supported.

Parameters:

vendor CSuite/Vendor

specifier CSuite/Specifier

Returns:

1 if ciphersuite is support, or 0 if not

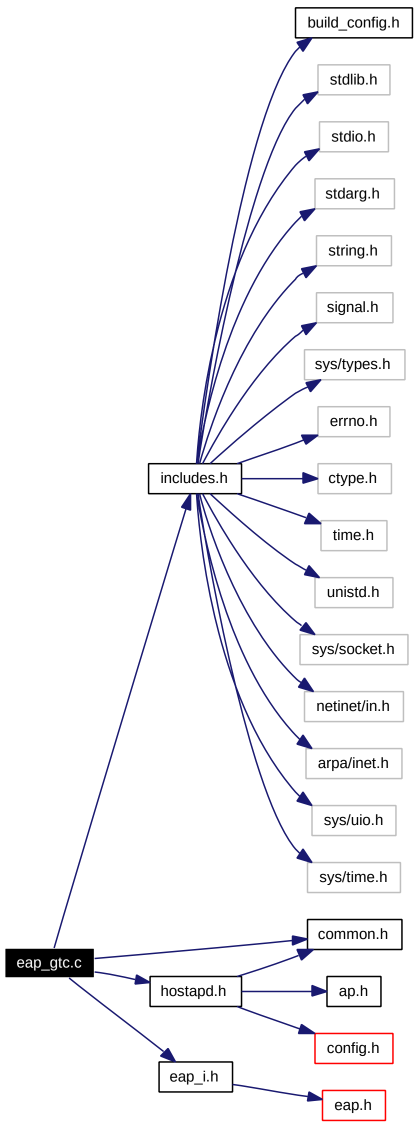
Definition at line 32 of file eap_gpsk_common.c.

6.38 eap_gtc.c File Reference

hostapd / EAP-GTC (RFC 3748)

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
```

Include dependency graph for eap_gtc.c:



Functions

- int **eap_server_gtc_register** (void)

6.38.1 Detailed Description

hostapd / EAP-GTC (RFC 3748)

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

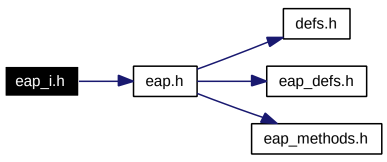
Definition in file eap_gtc.c.

6.39 eap_i.h File Reference

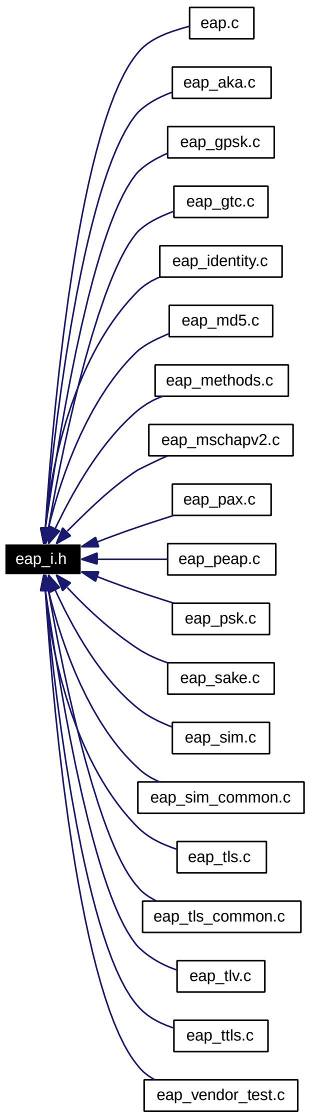
hostapd / EAP Authenticator state machine internal structures (RFC 4137)

`#include "eap.h"`

Include dependency graph for eap_i.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **EAP_SERVER_METHOD_INTERFACE_VERSION** 1

Functions

- int eap_user_get (struct eap_sm ∗sm, const u8 ∗identity, size_t identity_len, int phase2)

  Fetch user information from the database.

- void eap_sm_process_nak (struct eap_sm ∗sm, u8 ∗nak_list, size_t len)

  Process EAP-Response/Nak.

- const u8 ∗ eap_hdr_validate (int vendor, EapType eap_type, const u8 ∗msg, size_t msglen, size_t ∗plen)

  Validate EAP header.

- eap_hdr ∗ eap_msg_alloc (int vendor, EapType type, size_t ∗len, size_t payload_len, u8 code, u8 identifier, u8 ∗∗payload)

  Allocate a buffer for an EAP message.

6.39.1 Detailed Description

hostapd / EAP Authenticator state machine internal structures (RFC 4137)

Copyright

Definition in file eap_i.h.

6.39.2 Function Documentation

6.39.2.1 const u8∗ eap_hdr_validate (int *vendor*, EapType *eap_type*, const u8 ∗ *msg*, size_t *msglen*, size_t ∗ *plen*)

Validate EAP header.

Parameters:

vendor Expected EAP Vendor-Id (0 = IETF)

eap_type Expected EAP type number

msg EAP frame (starting with EAP header)

msglen Length of msg

plen Pointer to variable to contain the returned payload length

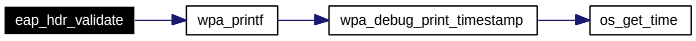
Returns:

Pointer to EAP payload (after type field), or NULL on failure

This is a helper function for EAP method implementations. This is usually called in the beginning of struct eap_method::process() function to verify that the received EAP request packet has a valid header. This function is able to process both legacy and expanded EAP headers and in most cases, the caller can just use the returned payload pointer (into ∗plen) for processing the payload regardless of whether the packet used the expanded EAP header or not.

Definition at line 1048 of file eap.c.

Here is the call graph for this function:



6.39.2.2 struct eap_hdr∗ eap_msg_alloc (int *vendor*, EapType *type*, size_t ∗ *len*, size_t *payload_len*, u8 *code*, u8 *identifier*, u8 ∗∗ *payload*)

Allocate a buffer for an EAP message.

Parameters:

vendor Vendor-Id (0 = IETF)

type EAP type

len Buffer for returning message length

payload_len Payload length in bytes (data after Type)

code Message Code (EAP_CODE_∗)

identifier Identifier

payload Pointer to payload pointer that will be set to point to the beginning of the payload or NULL if payload pointer is not needed

Returns:

Pointer to the allocated message buffer or NULL on error

This function can be used to allocate a buffer for an EAP message and fill in the EAP header. This function is automatically using expanded EAP header if the selected Vendor-Id is not IETF. In other words, most EAP methods do not need to separately select which header type to use when using this function to allocate the message buffers.

Definition at line 1121 of file eap.c.

6.39.2.3 void eap_sm_process_nak (struct eap_sm ∗ *sm*, u8 ∗ *nak_list*, size_t *len*)

Process EAP-Response/Nak.

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

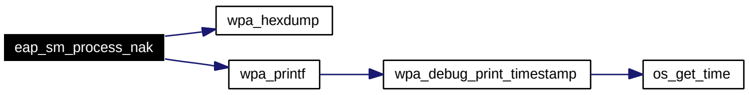
nak_list Nak list (allowed methods) from the supplicant

len Length of nak_list in bytes

This function is called when EAP-Response/Nak is received from the supplicant. This can happen for both phase 1 and phase 2 authentications.

Definition at line 714 of file eap.c.

Here is the call graph for this function:



6.39.2.4 int eap_user_get (struct eap_sm ∗ *sm*, const u8 ∗ *identity*, size_t *identity_len*, int *phase2*)

Fetch user information from the database.

Parameters:

sm Pointer to EAP state machine allocated with eap_sm_init()

identity Identity (User-Name) of the user

identity_len Length of identity in bytes

phase2 0 = EAP phase1 user, 1 = EAP phase2 (tunneled) user

Returns:

0 on success, or -1 on failure

This function is used to fetch user information for EAP. The user will be selected based on the specified identity. sm->user and sm->user_eap_method_index are updated for the new user when a matching user is found. sm->user can be used to get user information (e.g., password).

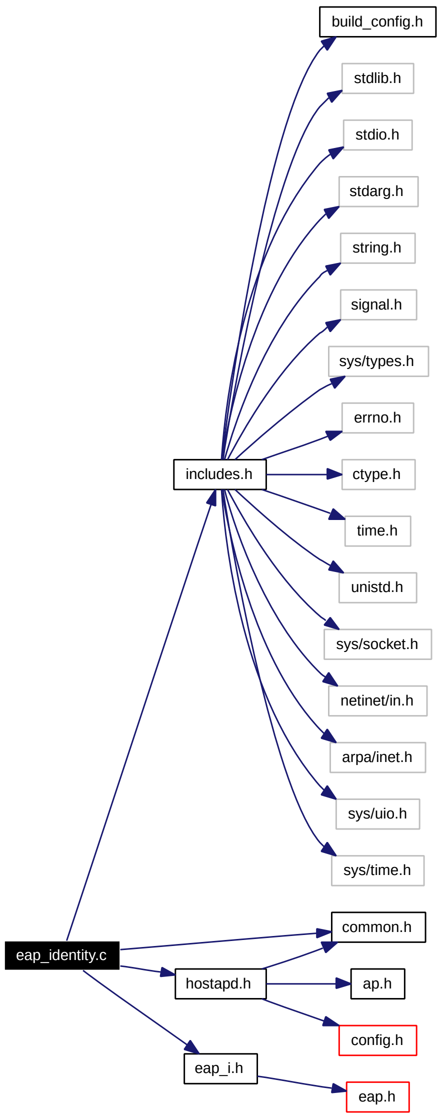
Definition at line 91 of file eap.c.

6.40 eap_identity.c File Reference

hostapd / EAP-Identity

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
```

Include dependency graph for eap_identity.c:



Functions

- int **eap_server_identity_register** (void)

6.40.1 Detailed Description

hostapd / EAP-Identity

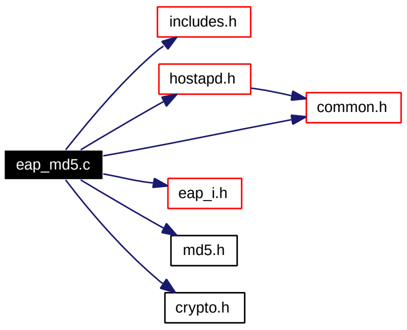
Definition in file eap_identity.c.

6.41 eap_md5.c File Reference

hostapd / EAP-MD5 server

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
#include "md5.h"
#include "crypto.h"
```

Include dependency graph for eap_md5.c:



Defines

- #define **CHALLENGE_LEN** 16

Functions

- int **eap_server_md5_register** (void)

6.41.1 Detailed Description

hostapd / EAP-MD5 server

Copyright
Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

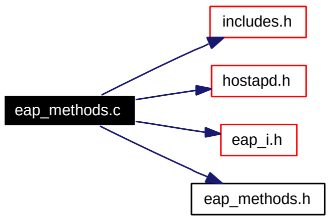
Definition in file eap_md5.c.

6.42 eap_methods.c File Reference

hostapd / EAP method registration

```
#include "includes.h"
#include "hostapd.h"
#include "eap_i.h"
#include "eap_methods.h"
```

Include dependency graph for eap_methods.c:



Functions

- const struct eap_method ∗ eap_sm_get_eap_methods (int vendor, EapType method)

  Get EAP method based on type number.

- EapType eap_get_type (const char ∗name, int ∗vendor)

  Get EAP type for the given EAP method name.

- eap_method ∗ eap_server_method_alloc (int version, int vendor, EapType method, const char ∗name)

  Allocate EAP server method structure.

- void eap_server_method_free (struct eap_method ∗method)

  Free EAP server method structure.

- int eap_server_method_register (struct eap_method ∗method)

  Register an EAP server method.

- int eap_server_register_methods (void)

  Register statically linked EAP server methods.

- void eap_server_unregister_methods (void)

  Unregister EAP server methods.

6.42.1 Detailed Description

hostapd / EAP method registration

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

Definition in file eap_methods.c.

6.42.2 Function Documentation

6.42.2.1 EapType eap_get_type (const char * *name*, int * *vendor*)

Get EAP type for the given EAP method name.

Parameters:

name EAP method name, e.g., TLS

vendor Buffer for returning EAP Vendor-Id

Returns:

EAP method type or EAP_TYPE_NONE if not found

This function maps EAP type names into EAP type numbers based on the list of EAP methods included in the build.

Definition at line 54 of file eap_methods.c.

6.42.2.2 struct eap_method* eap_server_method_alloc (int *version*, int *vendor*, EapType *method*, const char * *name*)

Allocate EAP server method structure.

Parameters:

version Version of the EAP server method interface (set to EAP_SERVER_METHOD_-INTERFACE_VERSION)

vendor EAP Vendor-ID (EAP_VENDOR_*) (0 = IETF)

method EAP type number (EAP_TYPE_*) name: Name of the method (e.g., "TLS")

Returns:

Allocated EAP method structure or NULL on failure

The returned structure should be freed with eap_server_method_free() when it is not needed anymore.

Definition at line 81 of file eap_methods.c.

6.42.2.3 void eap_server_method_free (struct eap_method * *method*)

Free EAP server method structure.

Parameters:

method Method structure allocated with eap_server_method_alloc()

Definition at line 101 of file eap_methods.c.

6.42.2.4 int eap_server_method_register (struct eap_method ∗ *method*)

Register an EAP server method.

Parameters:

method EAP method to register

Returns:

0 on success, -1 on invalid method, or -2 if a matching EAP method has already been registered

Each EAP server method needs to call this function to register itself as a supported EAP method.

Definition at line 117 of file eap_methods.c.

6.42.2.5 int eap_server_register_methods (void)

Register statically linked EAP server methods.

Returns:

0 on success, -1 on failure

This function is called at program initialization to register all EAP server methods that were linked in statically.

Definition at line 150 of file eap_methods.c.

6.42.2.6 void eap_server_unregister_methods (void)

Unregister EAP server methods.

This function is called at program termination to unregister all EAP server methods.

Definition at line 268 of file eap_methods.c.

Here is the call graph for this function:

eap_server_unregister_methods
eap_server_method_free

6.42.2.7 const struct eap_method∗ eap_sm_get_eap_methods (int *vendor*, EapType *method*)

Get EAP method based on type number.

Parameters:

vendor EAP Vendor-Id (0 = IETF)

method EAP type number

Returns:

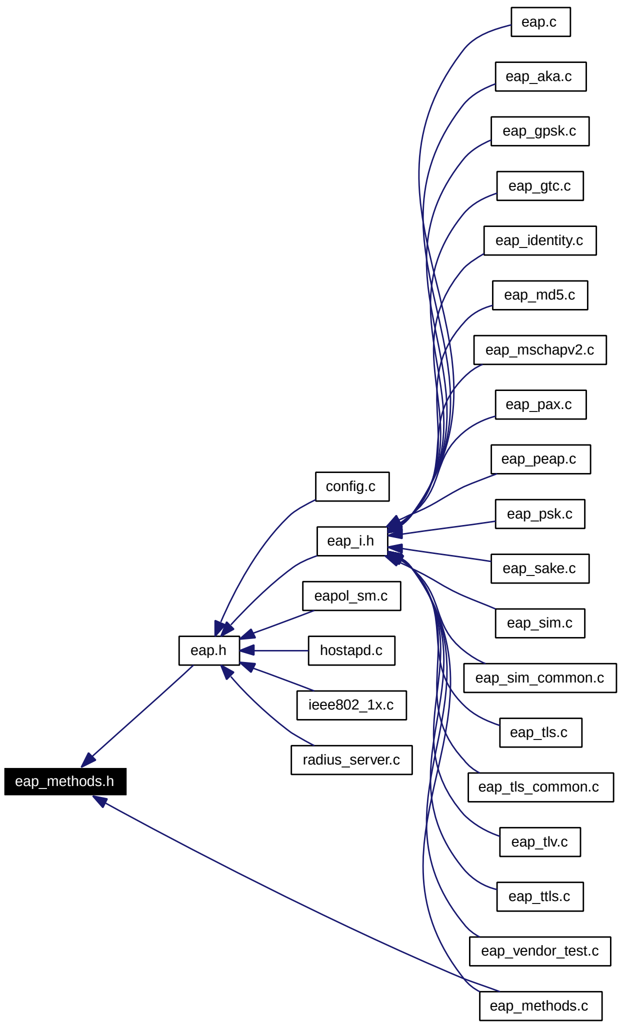
Pointer to EAP method or NULL if not found

Definition at line 33 of file eap_methods.c.

6.43 eap_methods.h File Reference

hostapd / EAP method registration

This graph shows which files directly or indirectly include this file:



Functions

- const struct eap_method ∗ eap_sm_get_eap_methods (int vendor, EapType method)

  Get EAP method based on type number.

- eap_method ∗ eap_server_method_alloc (int version, int vendor, EapType method, const char ∗name)

  Allocate EAP server method structure.

- void eap_server_method_free (struct eap_method ∗method)

  Free EAP server method structure.

- int eap_server_method_register (struct eap_method ∗method)

  Register an EAP server method.

- EapType eap_get_type (const char ∗name, int ∗vendor)

  Get EAP type for the given EAP method name.

- int eap_server_register_methods (void)

  Register statically linked EAP server methods.

- void eap_server_unregister_methods (void)

  Unregister EAP server methods.

6.43.1 Detailed Description

hostapd / EAP method registration

Copyright

Copyright (c) 2004-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eap_methods.h.

6.43.2 Function Documentation

6.43.2.1 EapType eap_get_type (const char ∗ *name*, int ∗ *vendor*)

Get EAP type for the given EAP method name.

Parameters:

name EAP method name, e.g., TLS

vendor Buffer for returning EAP Vendor-Id

Returns:

EAP method type or EAP_TYPE_NONE if not found

This function maps EAP type names into EAP type numbers based on the list of EAP methods included in the build.

Definition at line 54 of file eap_methods.c.

6.43.2.2 struct eap_method∗ eap_server_method_alloc (int *version*, int *vendor*, EapType *method*, const char ∗ *name*)

Allocate EAP server method structure.

Parameters:

version Version of the EAP server method interface (set to EAP_SERVER_METHOD_-INTERFACE_VERSION)

vendor EAP Vendor-ID (EAP_VENDOR_∗) (0 = IETF)

method EAP type number (EAP_TYPE_∗) name: Name of the method (e.g., "TLS")

Returns:

Allocated EAP method structure or NULL on failure

The returned structure should be freed with eap_server_method_free() when it is not needed anymore.

Definition at line 81 of file eap_methods.c.

6.43.2.3 void eap_server_method_free (struct eap_method ∗ *method*)

Free EAP server method structure.

Parameters:

method Method structure allocated with eap_server_method_alloc()

Definition at line 101 of file eap_methods.c.

6.43.2.4 int eap_server_method_register (struct eap_method ∗ *method*)

Register an EAP server method.

Parameters:

method EAP method to register

Returns:

0 on success, -1 on invalid method, or -2 if a matching EAP method has already been registered

Each EAP server method needs to call this function to register itself as a supported EAP method.

Definition at line 117 of file eap_methods.c.

6.43.2.5 int eap_server_register_methods (void)

Register statically linked EAP server methods.

Returns:

0 on success, -1 on failure

This function is called at program initialization to register all EAP server methods that were linked in statically.

Definition at line 150 of file eap_methods.c.

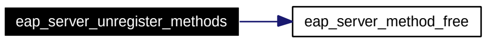
6.43.2.6 void eap_server_unregister_methods (void)

Unregister EAP server methods.

This function is called at program termination to unregister all EAP server methods.

Definition at line 268 of file eap_methods.c.

Here is the call graph for this function:



6.43.2.7 const struct eap_method* eap_sm_get_eap_methods (int *vendor*, EapType *method*)

Get EAP method based on type number.

Parameters:

vendor	EAP Vendor-Id (0 = IETF)
method	EAP type number

Returns:

Pointer to EAP method or NULL if not found

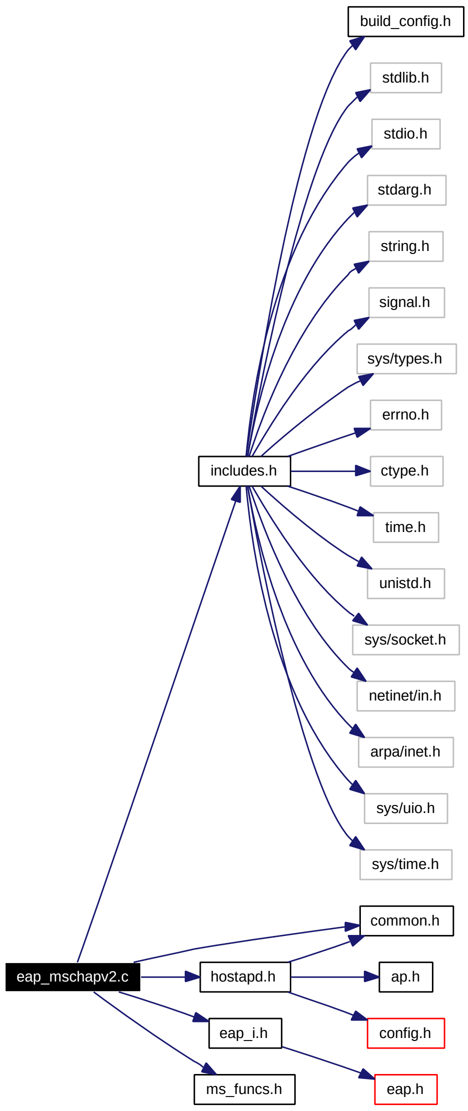
Definition at line 33 of file eap_methods.c.

6.44 eap_mschapv2.c File Reference

hostapd / EAP-MSCHAPv2 (draft-kamath-pppext-eap-mschapv2-00.txt) server

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
#include "ms_funcs.h"
```

Include dependency graph for eap_mschapv2.c:



Defines

- #define **MSCHAPV2_OP_CHALLENGE** 1

- #define **MSCHAPV2_OP_RESPONSE** 2
- #define **MSCHAPV2_OP_SUCCESS** 3
- #define **MSCHAPV2_OP_FAILURE** 4
- #define **MSCHAPV2_OP_CHANGE_PASSWORD** 7
- #define **MSCHAPV2_RESP_LEN** 49
- #define **ERROR_RESTRICTED_LOGON_HOURS** 646
- #define **ERROR_ACCT_DISABLED** 647
- #define **ERROR_PASSWD_EXPIRED** 648
- #define **ERROR_NO_DIALIN_PERMISSION** 649
- #define **ERROR_AUTHENTICATION_FAILURE** 691
- #define **ERROR_CHANGING_PASSWORD** 709
- #define **PASSWD_CHANGE_CHAL_LEN** 16
- #define **MSCHAPV2_KEY_LEN** 16
- #define **CHALLENGE_LEN** 16

Functions

- int **eap_server_mschapv2_register** (void)

Variables

- eap_mschapv2_hdr **STRUCT_PACKED**

6.44.1 Detailed Description

hostapd / EAP-MSCHAPv2 (draft-kamath-pppext-eap-mschapv2-00.txt) server

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eap_mschapv2.c.

6.45 eap_pax.c File Reference

hostapd / EAP-PAX (draft-clancy-eap-pax-11.txt) server

`#include "includes.h"`

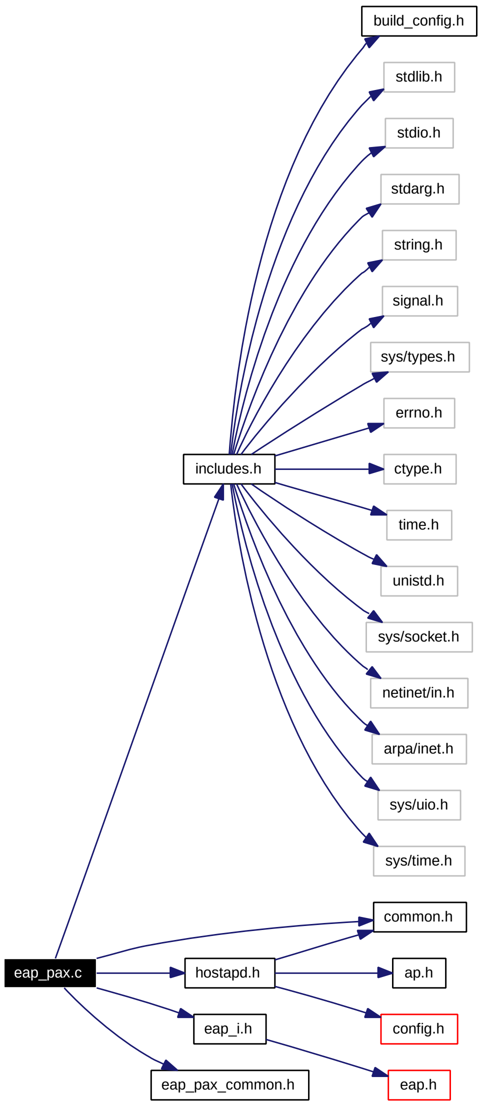
`#include "hostapd.h"`

`#include "common.h"`

`#include "eap_i.h"`

`#include "eap_pax_common.h"`

Include dependency graph for eap_pax.c:



Functions

- int **eap_server_pax_register** (void)

6.45.1 Detailed Description

hostapd / EAP-PAX (draft-clancy-eap-pax-11.txt) server

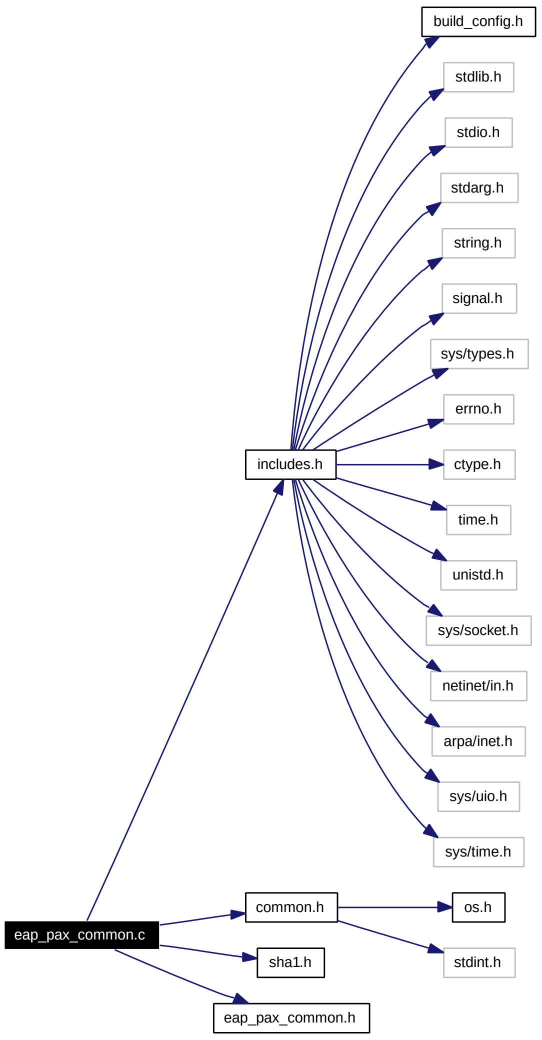
Definition in file eap_pax.c.

6.46 eap_pax_common.c File Reference

EAP server/peer: EAP-PAX shared routines.

```
#include "includes.h"
#include "common.h"
#include "sha1.h"
#include "eap_pax_common.h"
```

Include dependency graph for eap_pax_common.c:



Functions

- int eap_pax_kdf (u8 mac_id, const u8 *key, size_t key_len, const char *identifier, const u8 *entropy, size_t entropy_len, size_t output_len, u8 *output)

  PAX Key Derivation Function.

- int eap_pax_mac (u8 mac_id, const u8 ∗key, size_t key_len, const u8 ∗data1, size_t data1_len, const u8 ∗data2, size_t data2_len, const u8 ∗data3, size_t data3_len, u8 ∗mac)

  EAP-PAX MAC.

- int eap_pax_initial_key_derivation (u8 mac_id, const u8 ∗ak, const u8 ∗e, u8 ∗mk, u8 ∗ck, u8 ∗ick)

  EAP-PAX initial key derivation.

6.46.1 Detailed Description

EAP server/peer: EAP-PAX shared routines.

Copyright

Copyright (c) 2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eap_pax_common.c.

6.46.2 Function Documentation

6.46.2.1 int eap_pax_initial_key_derivation (u8 *mac_id*, const u8 ∗ *ak*, const u8 ∗ *e*, u8 ∗ *mk*, u8 ∗ *ck*, u8 ∗ *ick*)

EAP-PAX initial key derivation.

Parameters:

mac_id MAC ID (EAP_PAX_MAC_∗) / currently, only HMAC_SHA1_128 is supported

ak Authentication Key

e Entropy

mk Buffer for the derived Master Key

ck Buffer for the derived Confirmation Key

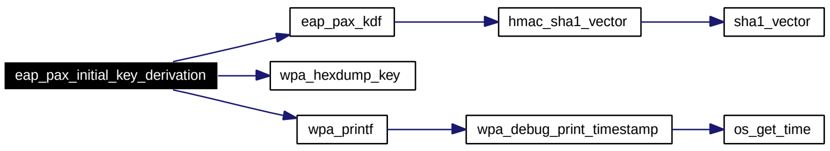
ick Buffer for the derived Integrity Check Key

Returns:

0 on success, -1 on failure

Definition at line 136 of file eap_pax_common.c.

Here is the call graph for this function:



6.46.2.2 int eap_pax_kdf (u8 *mac_id*, const u8 ∗ *key*, size_t *key_len*, const char ∗ *identifier*, const u8 ∗ *entropy*, size_t *entropy_len*, size_t *output_len*, u8 ∗ *output*)

PAX Key Derivation Function.

Parameters:

mac_id MAC ID (EAP_PAX_MAC_∗) / currently, only HMAC_SHA1_128 is supported

key Secret key (X)

key_len Length of the secret key in bytes

identifier Public identifier for the key (Y)

entropy Exchanged entropy to seed the KDF (Z)

entropy_len Length of the entropy in bytes

output_len Output len in bytes (W)

output Buffer for the derived key

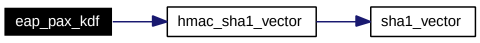
Returns:

0 on success, -1 failed

draft-clancy-eap-pax-04.txt, chap. 2.5: PAX-KDF-W(X, Y, Z)

Definition at line 38 of file eap_pax_common.c.

Here is the call graph for this function:



6.46.2.3 int eap_pax_mac (u8 *mac_id*, const u8 ∗ *key*, size_t *key_len*, const u8 ∗ *data1*, size_t *data1_len*, const u8 ∗ *data2*, size_t *data2_len*, const u8 ∗ *data3*, size_t *data3_len*, u8 ∗ *mac*)

EAP-PAX MAC.

Parameters:

mac_id MAC ID (EAP_PAX_MAC_∗) / currently, only HMAC_SHA1_128 is supported

key Secret key

key_len Length of the secret key in bytes

data1 Optional data, first block; NULL if not used

data1_len Length of data1 in bytes

data2 Optional data, second block; NULL if not used

data2_len Length of data2 in bytes

data3 Optional data, third block; NULL if not used

data3_len Length of data3 in bytes

mac Buffer for the MAC value (EAP_PAX_MAC_LEN = 16 bytes)

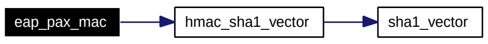
Returns:

0 on success, -1 on failure

Wrapper function to calculate EAP-PAX MAC.

Definition at line 95 of file eap_pax_common.c.

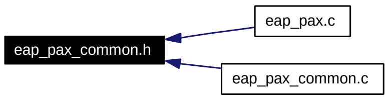
Here is the call graph for this function:



6.47 eap_pax_common.h File Reference

EAP server/peer: EAP-PAX shared routines.

This graph shows which files directly or indirectly include this file:



Defines

- #define **EAP_PAX_FLAGS_MF** 0x01
- #define **EAP_PAX_FLAGS_CE** 0x02
- #define **EAP_PAX_FLAGS_AI** 0x04
- #define **EAP_PAX_MAC_HMAC_SHA1_128** 0x01
- #define **EAP_PAX_HMAC_SHA256_128** 0x02
- #define **EAP_PAX_DH_GROUP_NONE** 0x00
- #define **EAP_PAX_DH_GROUP_2048_MODP** 0x01
- #define **EAP_PAX_DH_GROUP_3072_MODP** 0x02
- #define **EAP_PAX_DH_GROUP_NIST_ECC_P_256** 0x03
- #define **EAP_PAX_PUBLIC_KEY_NONE** 0x00
- #define **EAP_PAX_PUBLIC_KEY_RSAES_OAEP** 0x01
- #define **EAP_PAX_PUBLIC_KEY_RSA_PKCS1_V1_5** 0x02
- #define **EAP_PAX_PUBLIC_KEY_EL_GAMAL_NIST_ECC** 0x03
- #define **EAP_PAX_ADE_VENDOR_SPECIFIC** 0x01
- #define **EAP_PAX_ADE_CLIENT_CHANNEL_BINDING** 0x02
- #define **EAP_PAX_ADE_SERVER_CHANNEL_BINDING** 0x03
- #define **EAP_PAX_RAND_LEN** 32
- #define **EAP_PAX_MAC_LEN** 16
- #define **EAP_PAX_ICV_LEN** 16
- #define **EAP_PAX_AK_LEN** 16
- #define **EAP_PAX_MK_LEN** 16
- #define **EAP_PAX_CK_LEN** 16
- #define **EAP_PAX_ICK_LEN** 16

Enumerations

- enum {

  EAP_PAX_OP_STD_1 = 0x01, **EAP_PAX_OP_STD_2** = 0x02, **EAP_PAX_OP_STD_3** = 0x03, **EAP_PAX_OP_SEC_1** = 0x11,

  EAP_PAX_OP_SEC_2 = 0x12, **EAP_PAX_OP_SEC_3** = 0x13, **EAP_PAX_OP_SEC_4** = 0x14, **EAP_PAX_OP_SEC_5** = 0x15,

  EAP_PAX_OP_ACK = 0x21 }

Functions

- int eap_pax_kdf (u8 mac_id, const u8 ∗key, size_t key_len, const char ∗identifier, const u8 ∗entropy, size_t entropy_len, size_t output_len, u8 ∗output)

  PAX Key Derivation Function.

- int eap_pax_mac (u8 mac_id, const u8 ∗key, size_t key_len, const u8 ∗data1, size_t data1_len, const u8 ∗data2, size_t data2_len, const u8 ∗data3, size_t data3_len, u8 ∗mac)

  EAP-PAX MAC.

- int eap_pax_initial_key_derivation (u8 mac_id, const u8 ∗ak, const u8 ∗e, u8 ∗mk, u8 ∗ck, u8 ∗ick)

  EAP-PAX initial key derivation.

Variables

- eap_pax_hdr **STRUCT_PACKED**

6.47.1 Detailed Description

EAP server/peer: EAP-PAX shared routines.

Copyright

Copyright (c) 2005-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eap_pax_common.h.

6.47.2 Function Documentation

6.47.2.1 int eap_pax_initial_key_derivation (u8 *mac_id*, const u8 ∗ *ak*, const u8 ∗ *e*, u8 ∗ *mk*, u8 ∗ *ck*, u8 ∗ *ick*)

EAP-PAX initial key derivation.

Parameters:

mac_id MAC ID (EAP_PAX_MAC_∗) / currently, only HMAC_SHA1_128 is supported

ak Authentication Key

e Entropy

mk Buffer for the derived Master Key

ck Buffer for the derived Confirmation Key

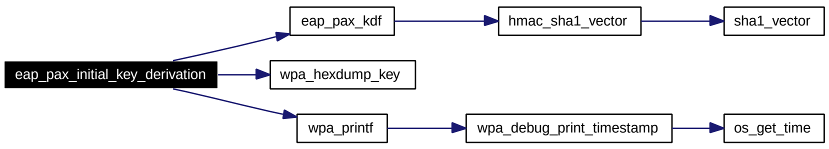
ick Buffer for the derived Integrity Check Key

Returns:

0 on success, -1 on failure

Definition at line 136 of file eap_pax_common.c.

Here is the call graph for this function:



6.47.2.2 int eap_pax_kdf (u8 *mac_id*, const u8 ∗ *key*, size_t *key_len*, const char ∗ *identifier*, const u8 ∗ *entropy*, size_t *entropy_len*, size_t *output_len*, u8 ∗ *output*)

PAX Key Derivation Function.

Parameters:

mac_id MAC ID (EAP_PAX_MAC_∗) / currently, only HMAC_SHA1_128 is supported

key Secret key (X)

key_len Length of the secret key in bytes

identifier Public identifier for the key (Y)

entropy Exchanged entropy to seed the KDF (Z)

entropy_len Length of the entropy in bytes

output_len Output len in bytes (W)

output Buffer for the derived key

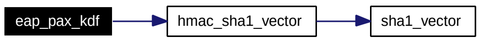
Returns:

0 on success, -1 failed

draft-clancy-eap-pax-04.txt, chap. 2.5: PAX-KDF-W(X, Y, Z)

Definition at line 38 of file eap_pax_common.c.

Here is the call graph for this function:



6.47.2.3 int eap_pax_mac (u8 *mac_id*, const u8 ∗ *key*, size_t *key_len*, const u8 ∗ *data1*, size_t *data1_len*, const u8 ∗ *data2*, size_t *data2_len*, const u8 ∗ *data3*, size_t *data3_len*, u8 ∗ *mac*)

EAP-PAX MAC.

Parameters:

mac_id MAC ID (EAP_PAX_MAC_∗) / currently, only HMAC_SHA1_128 is supported

key Secret key

key_len Length of the secret key in bytes

data1 Optional data, first block; NULL if not used

data1_len Length of data1 in bytes

data2 Optional data, second block; NULL if not used

data2_len Length of data2 in bytes

data3 Optional data, third block; NULL if not used

data3_len Length of data3 in bytes

mac Buffer for the MAC value (EAP_PAX_MAC_LEN = 16 bytes)

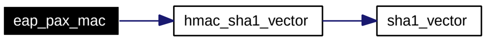
Returns:

0 on success, -1 on failure

Wrapper function to calculate EAP-PAX MAC.

Definition at line 95 of file eap_pax_common.c.

Here is the call graph for this function:

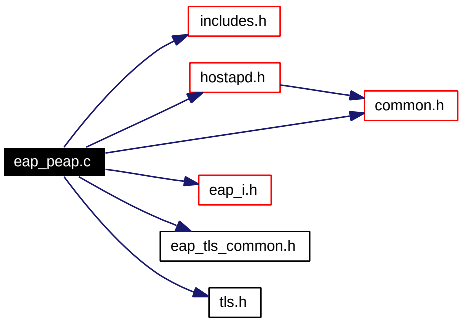


6.48 eap_peap.c File Reference

hostapd / EAP-PEAP (draft-josefsson-pppext-eap-tls-eap-07.txt)

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
#include "eap_tls_common.h"
#include "tls.h"
```

Include dependency graph for eap_peap.c:



Defines

- #define **EAP_PEAP_VERSION** 1

Functions

- int **eap_server_peap_register** (void)

6.48.1 Detailed Description

hostapd / EAP-PEAP (draft-josefsson-pppext-eap-tls-eap-07.txt)

Copyright
Copyright (c) 2004-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

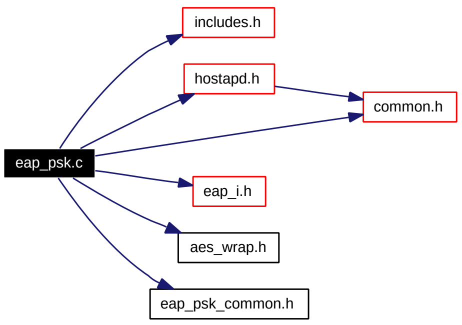
Definition in file eap_peap.c.

6.49 eap_psk.c File Reference

hostapd / EAP-PSK (draft-bersani-eap-psk-11.txt) server

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
#include "aes_wrap.h"
#include "eap_psk_common.h"
```

Include dependency graph for eap_psk.c:



Functions

- int **eap_server_psk_register** (void)

6.49.1 Detailed Description

hostapd / EAP-PSK (draft-bersani-eap-psk-11.txt) server

Copyright

Copyright (c) 2005-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Note: EAP-PSK is an EAP authentication method and as such, completely different from WPA-PSK. This file is not needed for WPA-PSK functionality.

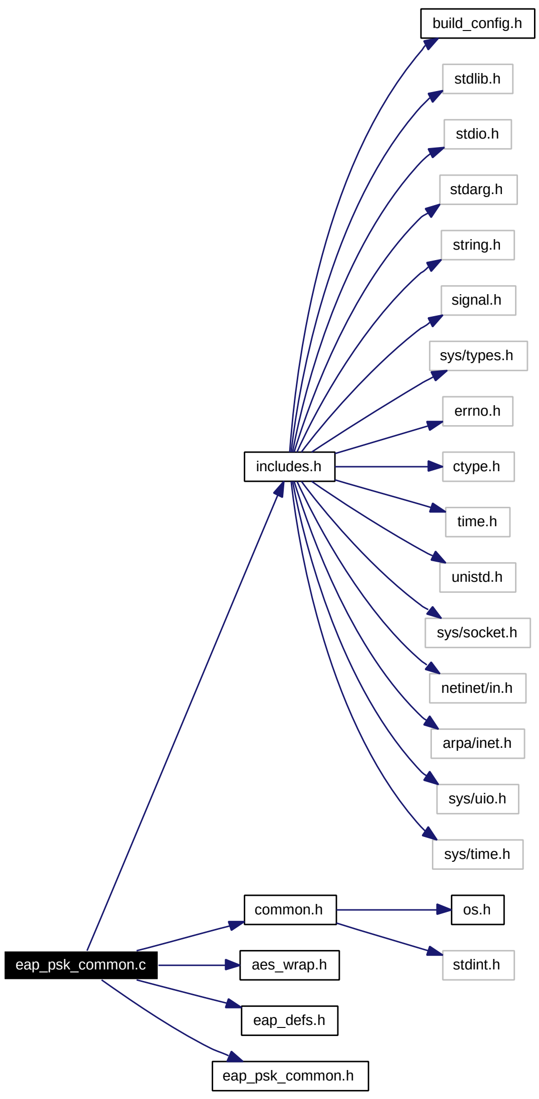
Definition in file eap_psk.c.

6.50 eap_psk_common.c File Reference

EAP server/peer: EAP-PSK shared routines.

```
#include "includes.h"
#include "common.h"
#include "aes_wrap.h"
#include "eap_defs.h"
#include "eap_psk_common.h"
```

Include dependency graph for eap_psk_common.c:



Defines

- #define **aes_block_size** 16

Functions

- void **eap_psk_key_setup** (const u8 ∗psk, u8 ∗ak, u8 ∗kdk)
- void **eap_psk_derive_keys** (const u8 ∗kdk, const u8 ∗rand_p, u8 ∗tek, u8 ∗msk, u8 ∗emsk)

6.50.1 Detailed Description

EAP server/peer: EAP-PSK shared routines.

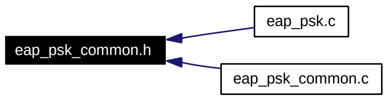
Copyright

Definition in file eap_psk_common.c.

6.51 eap_psk_common.h File Reference

EAP server/peer: EAP-PSK shared routines.

This graph shows which files directly or indirectly include this file:



Defines

- #define **EAP_PSK_RAND_LEN** 16
- #define **EAP_PSK_MAC_LEN** 16
- #define **EAP_PSK_TEK_LEN** 16
- #define **EAP_PSK_PSK_LEN** 16
- #define **EAP_PSK_AK_LEN** 16
- #define **EAP_PSK_KDK_LEN** 16
- #define **EAP_PSK_R_FLAG_CONT** 1
- #define **EAP_PSK_R_FLAG_DONE_SUCCESS** 2
- #define **EAP_PSK_R_FLAG_DONE_FAILURE** 3
- #define **EAP_PSK_E_FLAG** 0x20
- #define **EAP_PSK_FLAGS_GET_T**(flags) (((flags) & 0xc0) >> 6)
- #define **EAP_PSK_FLAGS_SET_T**(t) ((u8) (t) << 6)

Functions

- void **eap_psk_key_setup** (const u8 ∗psk, u8 ∗ak, u8 ∗kdk)
- void **eap_psk_derive_keys** (const u8 ∗kdk, const u8 ∗rand_p, u8 ∗tek, u8 ∗msk, u8 ∗emsk)

Variables

- eap_psk_hdr **STRUCT_PACKED**

6.51.1 Detailed Description

EAP server/peer: EAP-PSK shared routines.

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

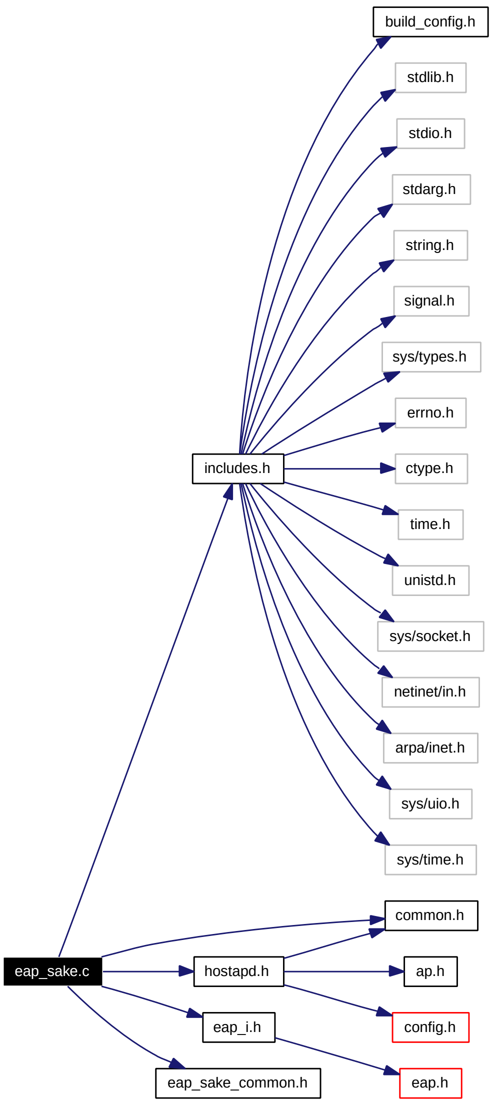
Definition in file eap_psk_common.h.

6.52 eap_sake.c File Reference

hostapd / EAP-SAKE (RFC 4763) server

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
#include "eap_sake_common.h"
```

Include dependency graph for eap_sake.c:



Functions

- int **eap_server_sake_register** (void)

6.52.1 Detailed Description

hostapd / EAP-SAKE (RFC 4763) server

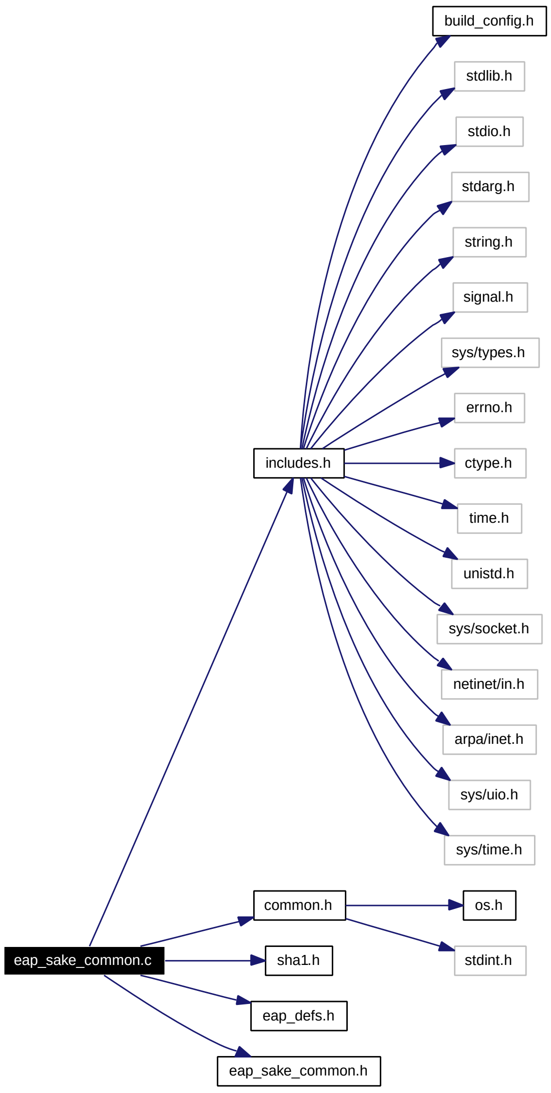
Definition in file eap_sake.c.

6.53 eap_sake_common.c File Reference

EAP server/peer: EAP-SAKE shared routines.

```
#include "includes.h"
#include "common.h"
#include "sha1.h"
#include "eap_defs.h"
#include "eap_sake_common.h"
```

Include dependency graph for eap_sake_common.c:



Functions

- int eap_sake_parse_attributes (const u8 ∗buf, size_t len, struct eap_sake_parse_attr ∗attr)

Parse EAP-SAKE attributes.

- void eap_sake_derive_keys (const u8 ∗root_secret_a, const u8 ∗root_secret_b, const u8 ∗rand_s, const u8 ∗rand_p, u8 ∗tek, u8 ∗msk, u8 ∗emsk)

  Derive EAP-SAKE keys.

- int eap_sake_compute_mic (const u8 ∗tek_auth, const u8 ∗rand_s, const u8 ∗rand_p, const u8 ∗serverid, size_t serverid_len, const u8 ∗peerid, size_t peerid_len, int peer, const u8 ∗eap, size_t eap_len, const u8 ∗mic_pos, u8 ∗mic)

  Compute EAP-SAKE MIC for an EAP packet.

6.53.1 Detailed Description

EAP server/peer: EAP-SAKE shared routines.

Copyright

Copyright (c) 2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eap_sake_common.c.

6.53.2 Function Documentation

6.53.2.1 int eap_sake_compute_mic (const u8 ∗ *tek_auth*, const u8 ∗ *rand_s*, const u8 ∗ *rand_p*, const u8 ∗ *serverid*, size_t *serverid_len*, const u8 ∗ *peerid*, size_t *peerid_len*, int *peer*, const u8 ∗ *eap*, size_t *eap_len*, const u8 ∗ *mic_pos*, u8 ∗ *mic*)

Compute EAP-SAKE MIC for an EAP packet.

Parameters:

tek_auth	16-byte TEK-Auth
rand_s	16-byte RAND_S
rand_p	16-byte RAND_P
serverid	SERVERID
serverid_len	SERVERID length
peerid	PEERID
peerid_len	PEERID length
peer	MIC calculation for 0 = Server, 1 = Peer message
eap	EAP packet
eap_len	EAP packet length
mic_pos	MIC position in the EAP packet (must be [eap .. eap + eap_len])
mic	Buffer for the computed 16-byte MIC

Definition at line 326 of file eap_sake_common.c.

6.53.2.2 void eap_sake_derive_keys (const u8 ∗ *root_secret_a*, const u8 ∗ *root_secret_b*, const u8 ∗ *rand_s*, const u8 ∗ *rand_p*, u8 ∗ *tek*, u8 ∗ *msk*, u8 ∗ *emsk*)

Derive EAP-SAKE keys.

Parameters:

root_secret_a 16-byte Root-Secret-A

root_secret_b 16-byte Root-Secret-B

rand_s 16-byte RAND_S

rand_p 16-byte RAND_P

tek Buffer for Temporary EAK Keys (TEK-Auth[16] | TEK-Cipher[16])

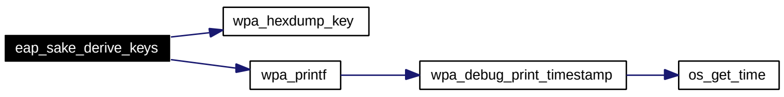
msk Buffer for 64-byte MSK

emsk Buffer for 64-byte EMSK

This function derives EAP-SAKE keys as defined in RFC 4763, section 3.2.6.

Definition at line 268 of file eap_sake_common.c.

Here is the call graph for this function:



6.53.2.3 int eap_sake_parse_attributes (const u8 ∗ *buf*, size_t *len*, struct eap_sake_parse_attr ∗ *attr*)

Parse EAP-SAKE attributes.

Parameters:

buf Packet payload (starting with the first attribute)

len Payload length

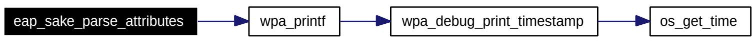
attr Structure to be filled with found attributes

Returns:

0 on success or -1 on failure

Definition at line 167 of file eap_sake_common.c.

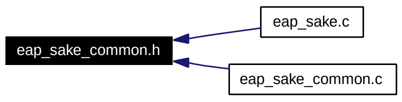
Here is the call graph for this function:



6.54 eap_sake_common.h File Reference

EAP server/peer: EAP-SAKE shared routines.

This graph shows which files directly or indirectly include this file:



Defines

- #define **EAP_SAKE_VERSION** 2
- #define **EAP_SAKE_SUBTYPE_CHALLENGE** 1
- #define **EAP_SAKE_SUBTYPE_CONFIRM** 2
- #define **EAP_SAKE_SUBTYPE_AUTH_REJECT** 3
- #define **EAP_SAKE_SUBTYPE_IDENTITY** 4
- #define **EAP_SAKE_AT_RAND_S** 1
- #define **EAP_SAKE_AT_RAND_P** 2
- #define **EAP_SAKE_AT_MIC_S** 3
- #define **EAP_SAKE_AT_MIC_P** 4
- #define **EAP_SAKE_AT_SERVERID** 5
- #define **EAP_SAKE_AT_PEERID** 6
- #define **EAP_SAKE_AT_SPI_S** 7
- #define **EAP_SAKE_AT_SPI_P** 8
- #define **EAP_SAKE_AT_ANY_ID_REQ** 9
- #define **EAP_SAKE_AT_PERM_ID_REQ** 10
- #define **EAP_SAKE_AT_ENCR_DATA** 128
- #define **EAP_SAKE_AT_IV** 129
- #define **EAP_SAKE_AT_PADDING** 130
- #define **EAP_SAKE_AT_NEXT_TMPID** 131
- #define **EAP_SAKE_AT_MSK_LIFE** 132
- #define **EAP_SAKE_RAND_LEN** 16
- #define **EAP_SAKE_MIC_LEN** 16
- #define **EAP_SAKE_ROOT_SECRET_LEN** 16
- #define **EAP_SAKE_SMS_LEN** 16
- #define **EAP_SAKE_TEK_AUTH_LEN** 16
- #define **EAP_SAKE_TEK_CIPHER_LEN** 16
- #define **EAP_SAKE_TEK_LEN** (EAP_SAKE_TEK_AUTH_LEN + EAP_SAKE_TEK_-CIPHER_LEN)

Functions

- int eap_sake_parse_attributes (const u8 ∗buf, size_t len, struct eap_sake_parse_attr ∗attr)

  Parse EAP-SAKE attributes.

- void eap_sake_derive_keys (const u8 ∗root_secret_a, const u8 ∗root_secret_b, const u8 ∗rand_s, const u8 ∗rand_p, u8 ∗tek, u8 ∗msk, u8 ∗emsk)

  Derive EAP-SAKE keys.

- int eap_sake_compute_mic (const u8 ∗tek_auth, const u8 ∗rand_s, const u8 ∗rand_p, const u8 ∗serverid, size_t serverid_len, const u8 ∗peerid, size_t peerid_len, int peer, const u8 ∗eap, size_t eap_len, const u8 ∗mic_pos, u8 ∗mic)

  Compute EAP-SAKE MIC for an EAP packet.

Variables

- eap_sake_hdr **STRUCT_PACKED**

6.54.1 Detailed Description

EAP server/peer: EAP-SAKE shared routines.

Copyright

Copyright (c) 2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eap_sake_common.h.

6.54.2 Function Documentation

6.54.2.1 int eap_sake_compute_mic (const u8 ∗ *tek_auth*, const u8 ∗ *rand_s*, const u8 ∗ *rand_p*, const u8 ∗ *serverid*, size_t *serverid_len*, const u8 ∗ *peerid*, size_t *peerid_len*, int *peer*, const u8 ∗ *eap*, size_t *eap_len*, const u8 ∗ *mic_pos*, u8 ∗ *mic*)

Compute EAP-SAKE MIC for an EAP packet.

Parameters:

tek_auth 16-byte TEK-Auth

rand_s 16-byte RAND_S

rand_p 16-byte RAND_P

serverid SERVERID

serverid_len SERVERID length

peerid PEERID

peerid_len PEERID length

peer MIC calculation for 0 = Server, 1 = Peer message

eap EAP packet

eap_len EAP packet length

mic_pos MIC position in the EAP packet (must be [eap .. eap + eap_len])

mic Buffer for the computed 16-byte MIC

Definition at line 326 of file eap_sake_common.c.

6.54.2.2 void eap_sake_derive_keys (const u8 ∗ *root_secret_a*, const u8 ∗ *root_secret_b*, const u8 ∗ *rand_s*, const u8 ∗ *rand_p*, u8 ∗ *tek*, u8 ∗ *msk*, u8 ∗ *emsk*)

Derive EAP-SAKE keys.

Parameters:

root_secret_a 16-byte Root-Secret-A

root_secret_b 16-byte Root-Secret-B

rand_s 16-byte RAND_S

rand_p 16-byte RAND_P

tek Buffer for Temporary EAK Keys (TEK-Auth[16] | TEK-Cipher[16])

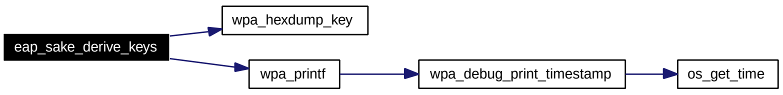
msk Buffer for 64-byte MSK

emsk Buffer for 64-byte EMSK

This function derives EAP-SAKE keys as defined in RFC 4763, section 3.2.6.

Definition at line 268 of file eap_sake_common.c.

Here is the call graph for this function:



6.54.2.3 int eap_sake_parse_attributes (const u8 ∗ *buf*, size_t *len*, struct eap_sake_parse_attr ∗ *attr*)

Parse EAP-SAKE attributes.

Parameters:

buf Packet payload (starting with the first attribute)

len Payload length

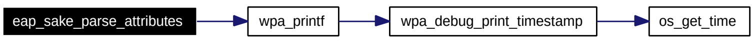
attr Structure to be filled with found attributes

Returns:

0 on success or -1 on failure

Definition at line 167 of file eap_sake_common.c.

Here is the call graph for this function:



6.55 eap_sim.c File Reference

hostapd / EAP-SIM (RFC 4186)

`#include "includes.h"`

`#include "hostapd.h"`

`#include "common.h"`

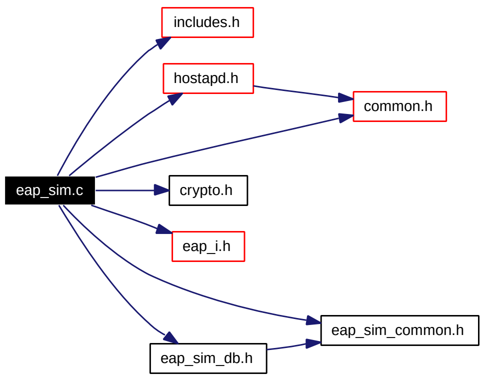
`#include "crypto.h"`

`#include "eap_i.h"`

`#include "eap_sim_common.h"`

`#include "eap_sim_db.h"`

Include dependency graph for eap_sim.c:



Functions

- int **eap_server_sim_register** (void)

6.55.1 Detailed Description

hostapd / EAP-SIM (RFC 4186)

Copyright

Copyright (c) 2005-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

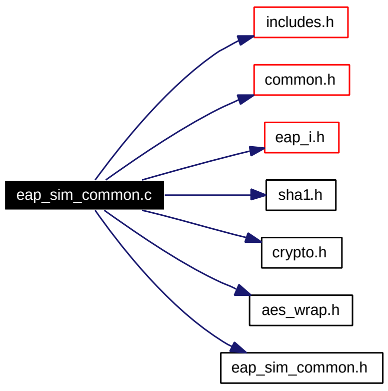
Definition in file eap_sim.c.

6.56 eap_sim_common.c File Reference

EAP peer: EAP-SIM/AKA shared routines.

```
#include "includes.h"
#include "common.h"
#include "eap_i.h"
#include "sha1.h"
#include "crypto.h"
#include "aes_wrap.h"
#include "eap_sim_common.h"
```

Include dependency graph for eap_sim_common.c:



Defines

- #define **EAP_SIM_INIT_LEN** 128

Functions

- void **eap_sim_derive_mk** (const u8 ∗identity, size_t identity_len, const u8 ∗nonce_mt, u16 selected_version, const u8 ∗ver_list, size_t ver_list_len, int num_chal, const u8 ∗kc, u8 ∗mk)
- void **eap_aka_derive_mk** (const u8 ∗identity, size_t identity_len, const u8 ∗ik, const u8 ∗ck, u8 ∗mk)
- int **eap_sim_derive_keys** (const u8 ∗mk, u8 ∗k_encr, u8 ∗k_aut, u8 ∗msk, u8 ∗emsk)
- int **eap_sim_derive_keys_reauth** (u16 _counter, const u8 ∗identity, size_t identity_len, const u8 ∗nonce_s, const u8 ∗mk, u8 ∗msk, u8 ∗emsk)
- int **eap_sim_verify_mac** (const u8 ∗k_aut, const u8 ∗req, size_t req_len, const u8 ∗mac, const u8 ∗extra, size_t extra_len)
- void **eap_sim_add_mac** (const u8 ∗k_aut, u8 ∗msg, size_t msg_len, u8 ∗mac, const u8 ∗extra, size_t extra_len)
- int **eap_sim_parse_attr** (const u8 ∗start, const u8 ∗end, struct eap_sim_attrs ∗attr, int aka, int encr)
- u8 ∗ **eap_sim_parse_encr** (const u8 ∗k_encr, const u8 ∗encr_data, size_t encr_data_len, const u8 ∗iv, struct eap_sim_attrs ∗attr, int aka)
- eap_sim_msg ∗ **eap_sim_msg_init** (int code, int id, int type, int subtype)

- u8 ∗ **eap_sim_msg_finish** (struct eap_sim_msg ∗msg, size_t ∗len, const u8 ∗k_aut, const u8 ∗extra, size_t extra_len)
- void **eap_sim_msg_free** (struct eap_sim_msg ∗msg)
- u8 ∗ **eap_sim_msg_add_full** (struct eap_sim_msg ∗msg, u8 attr, const u8 ∗data, size_t len)
- u8 ∗ **eap_sim_msg_add** (struct eap_sim_msg ∗msg, u8 attr, u16 value, const u8 ∗data, size_t len)
- u8 ∗ **eap_sim_msg_add_mac** (struct eap_sim_msg ∗msg, u8 attr)
- int **eap_sim_msg_add_encr_start** (struct eap_sim_msg ∗msg, u8 attr_iv, u8 attr_encr)
- int **eap_sim_msg_add_encr_end** (struct eap_sim_msg ∗msg, u8 ∗k_encr, int attr_pad)
- void **eap_sim_report_notification** (void ∗msg_ctx, int notification, int aka)

6.56.1 Detailed Description

EAP peer: EAP-SIM/AKA shared routines.

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

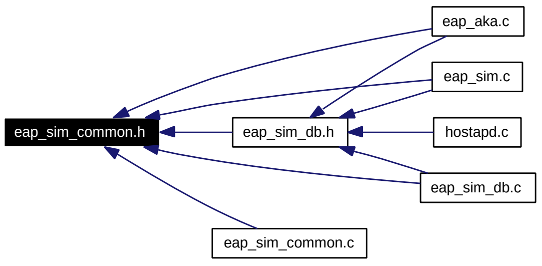
See README and COPYING for more details.

Definition in file eap_sim_common.c.

6.57 eap_sim_common.h File Reference

EAP peer: EAP-SIM/AKA shared routines.

This graph shows which files directly or indirectly include this file:



Defines

- #define **EAP_SIM_NONCE_S_LEN** 16
- #define **EAP_SIM_NONCE_MT_LEN** 16
- #define **EAP_SIM_MAC_LEN** 16
- #define **EAP_SIM_MK_LEN** 20
- #define **EAP_SIM_K_AUT_LEN** 16
- #define **EAP_SIM_K_ENCR_LEN** 16
- #define **EAP_SIM_KEYING_DATA_LEN** 64
- #define **EAP_SIM_IV_LEN** 16
- #define **EAP_SIM_KC_LEN** 8
- #define **EAP_SIM_SRES_LEN** 4
- #define **GSM_RAND_LEN** 16
- #define **EAP_SIM_VERSION** 1
- #define **EAP_SIM_SUBTYPE_START** 10
- #define **EAP_SIM_SUBTYPE_CHALLENGE** 11
- #define **EAP_SIM_SUBTYPE_NOTIFICATION** 12
- #define **EAP_SIM_SUBTYPE_REAUTHENTICATION** 13
- #define **EAP_SIM_SUBTYPE_CLIENT_ERROR** 14
- #define **EAP_SIM_UNABLE_TO_PROCESS_PACKET** 0
- #define **EAP_SIM_UNSUPPORTED_VERSION** 1
- #define **EAP_SIM_INSUFFICIENT_NUM_OF_CHAL** 2
- #define **EAP_SIM_RAND_NOT_FRESH** 3
- #define **EAP_SIM_MAX_FAST_REAUTHS** 1000
- #define **EAP_SIM_MAX_CHAL** 3
- #define **EAP_AKA_SUBTYPE_CHALLENGE** 1
- #define **EAP_AKA_SUBTYPE_AUTHENTICATION_REJECT** 2
- #define **EAP_AKA_SUBTYPE_SYNCHRONIZATION_FAILURE** 4
- #define **EAP_AKA_SUBTYPE_IDENTITY** 5
- #define **EAP_AKA_SUBTYPE_NOTIFICATION** 12
- #define **EAP_AKA_SUBTYPE_REAUTHENTICATION** 13
- #define **EAP_AKA_SUBTYPE_CLIENT_ERROR** 14
- #define **EAP_AKA_UNABLE_TO_PROCESS_PACKET** 0
- #define **EAP_AKA_RAND_LEN** 16

- #define **EAP_AKA_AUTN_LEN** 16
- #define **EAP_AKA_AUTS_LEN** 14
- #define **EAP_AKA_RES_MAX_LEN** 16
- #define **EAP_AKA_IK_LEN** 16
- #define **EAP_AKA_CK_LEN** 16
- #define **EAP_AKA_MAX_FAST_REAUTHS** 1000
- #define **EAP_AKA_MIN_RES_LEN** 4
- #define **EAP_AKA_MAX_RES_LEN** 16
- #define **EAP_SIM_AT_RAND** 1
- #define **EAP_SIM_AT_AUTN** 2
- #define **EAP_SIM_AT_RES** 3
- #define **EAP_SIM_AT_AUTS** 4
- #define **EAP_SIM_AT_PADDING** 6
- #define **EAP_SIM_AT_NONCE_MT** 7
- #define **EAP_SIM_AT_PERMANENT_ID_REQ** 10
- #define **EAP_SIM_AT_MAC** 11
- #define **EAP_SIM_AT_NOTIFICATION** 12
- #define **EAP_SIM_AT_ANY_ID_REQ** 13
- #define **EAP_SIM_AT_IDENTITY** 14
- #define **EAP_SIM_AT_VERSION_LIST** 15
- #define **EAP_SIM_AT_SELECTED_VERSION** 16
- #define **EAP_SIM_AT_FULLAUTH_ID_REQ** 17
- #define **EAP_SIM_AT_COUNTER** 19
- #define **EAP_SIM_AT_COUNTER_TOO_SMALL** 20
- #define **EAP_SIM_AT_NONCE_S** 21
- #define **EAP_SIM_AT_CLIENT_ERROR_CODE** 22
- #define **EAP_SIM_AT_IV** 129
- #define **EAP_SIM_AT_ENCR_DATA** 130
- #define **EAP_SIM_AT_NEXT_PSEUDONYM** 132
- #define **EAP_SIM_AT_NEXT_REAUTH_ID** 133
- #define **EAP_SIM_AT_CHECKCODE** 134
- #define **EAP_SIM_AT_RESULT_IND** 135
- #define **EAP_SIM_GENERAL_FAILURE_AFTER_AUTH** 0
- #define **EAP_SIM_TEMPORARILY_DENIED** 1026
- #define **EAP_SIM_NOT_SUBSCRIBED** 1031
- #define **EAP_SIM_GENERAL_FAILURE_BEFORE_AUTH** 16384
- #define **EAP_SIM_SUCCESS** 32768

Enumerations

- enum **eap_sim_id_req** { **NO_ID_REQ**, **ANY_ID**, **FULLAUTH_ID**, **PERMANENT_ID** }

Functions

- void **eap_sim_derive_mk** (const u8 ∗identity, size_t identity_len, const u8 ∗nonce_mt, u16 selected_version, const u8 ∗ver_list, size_t ver_list_len, int num_chal, const u8 ∗kc, u8 ∗mk)
- void **eap_aka_derive_mk** (const u8 ∗identity, size_t identity_len, const u8 ∗ik, const u8 ∗ck, u8 ∗mk)
- int **eap_sim_derive_keys** (const u8 ∗mk, u8 ∗k_encr, u8 ∗k_aut, u8 ∗msk, u8 ∗emsk)

- int **eap_sim_derive_keys_reauth** (u16 _counter, const u8 ∗identity, size_t identity_len, const u8 ∗nonce_s, const u8 ∗mk, u8 ∗msk, u8 ∗emsk)
- int **eap_sim_verify_mac** (const u8 ∗k_aut, const u8 ∗req, size_t req_len, const u8 ∗mac, const u8 ∗extra, size_t extra_len)
- void **eap_sim_add_mac** (const u8 ∗k_aut, u8 ∗msg, size_t msg_len, u8 ∗mac, const u8 ∗extra, size_t extra_len)
- int **eap_sim_parse_attr** (const u8 ∗start, const u8 ∗end, struct eap_sim_attrs ∗attr, int aka, int encr)
- u8 ∗ **eap_sim_parse_encr** (const u8 ∗k_encr, const u8 ∗encr_data, size_t encr_data_len, const u8 ∗iv, struct eap_sim_attrs ∗attr, int aka)
- eap_sim_msg ∗ **eap_sim_msg_init** (int code, int id, int type, int subtype)
- u8 ∗ **eap_sim_msg_finish** (struct eap_sim_msg ∗msg, size_t ∗len, const u8 ∗k_aut, const u8 ∗extra, size_t extra_len)
- void **eap_sim_msg_free** (struct eap_sim_msg ∗msg)
- u8 ∗ **eap_sim_msg_add_full** (struct eap_sim_msg ∗msg, u8 attr, const u8 ∗data, size_t len)
- u8 ∗ **eap_sim_msg_add** (struct eap_sim_msg ∗msg, u8 attr, u16 value, const u8 ∗data, size_t len)
- u8 ∗ **eap_sim_msg_add_mac** (struct eap_sim_msg ∗msg, u8 attr)
- int **eap_sim_msg_add_encr_start** (struct eap_sim_msg ∗msg, u8 attr_iv, u8 attr_encr)
- int **eap_sim_msg_add_encr_end** (struct eap_sim_msg ∗msg, u8 ∗k_encr, int attr_pad)
- void **eap_sim_report_notification** (void ∗msg_ctx, int notification, int aka)

6.57.1 Detailed Description

EAP peer: EAP-SIM/AKA shared routines.

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

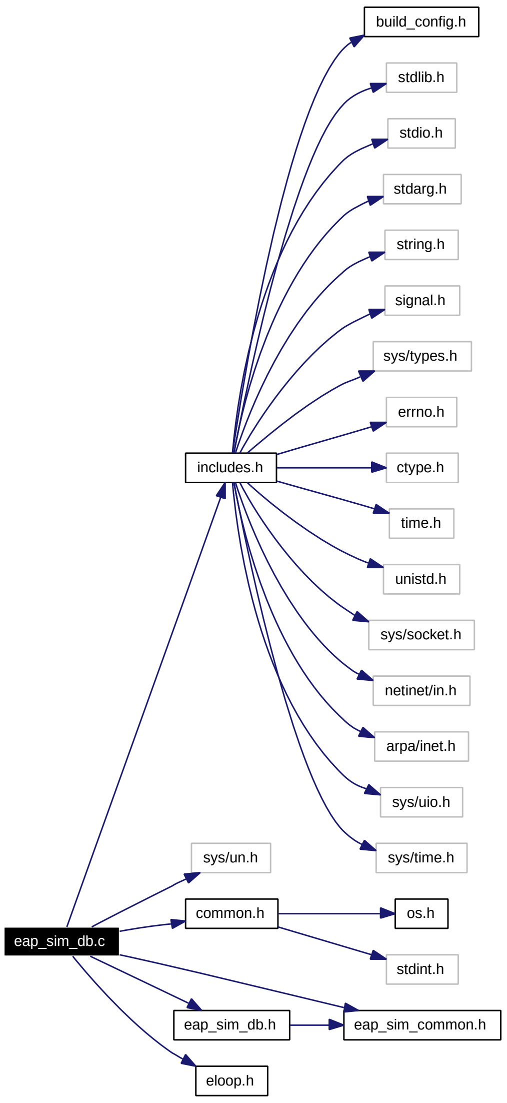
Definition in file eap_sim_common.h.

6.58 eap_sim_db.c File Reference

hostapd / EAP-SIM database/authenticator gateway

```
#include "includes.h"
#include <sys/un.h>
#include "common.h"
#include "eap_sim_common.h"
#include "eap_sim_db.h"
#include "eloop.h"
```

Include dependency graph for eap_sim_db.c:



Functions

- void ∗ eap_sim_db_init (const char ∗config, void(∗get_complete_cb)(void ∗ctx, void ∗session_ctx), void ∗ctx)

  Initialize EAP-SIM DB / authentication gateway interface.

- void eap_sim_db_deinit (void ∗priv)

  Deinitialize EAP-SIM DB/authentication gw interface.

- int eap_sim_db_get_gsm_triplets (void ∗priv, const u8 ∗identity, size_t identity_len, int max_chal, u8 ∗_rand, u8 ∗kc, u8 ∗sres, void ∗cb_session_ctx)

  Get GSM triplets.

- int eap_sim_db_identity_known (void ∗priv, const u8 ∗identity, size_t identity_len)

  Verify whether the given identity is known.

- char ∗ eap_sim_db_get_next_pseudonym (void ∗priv, int aka)

  EAP-SIM DB: Get next pseudonym.

- char ∗ eap_sim_db_get_next_reauth_id (void ∗priv, int aka)

  EAP-SIM DB: Get next reauth_id.

- int eap_sim_db_add_pseudonym (void ∗priv, const u8 ∗identity, size_t identity_len, char ∗pseudonym)

  EAP-SIM DB: Add new pseudonym.

- int eap_sim_db_add_reauth (void ∗priv, const u8 ∗identity, size_t identity_len, char ∗reauth_id, u16 counter, const u8 ∗mk)

  EAP-SIM DB: Add new re-authentication entry.

- const u8 ∗ eap_sim_db_get_permanent (void ∗priv, const u8 ∗identity, size_t identity_len, size_t ∗len)

  EAP-SIM DB: Get permanent identity.

- eap_sim_reauth ∗ eap_sim_db_get_reauth_entry (void ∗priv, const u8 ∗identity, size_t identity_-len)

  EAP-SIM DB: Get re-authentication entry.

- void eap_sim_db_remove_reauth (void ∗priv, struct eap_sim_reauth ∗reauth)

  EAP-SIM DB: Remove re-authentication entry.

- int eap_sim_db_get_aka_auth (void ∗priv, const u8 ∗identity, size_t identity_len, u8 ∗_rand, u8 ∗autn, u8 ∗ik, u8 ∗ck, u8 ∗res, size_t ∗res_len, void ∗cb_session_ctx)

  Get AKA authentication values.

- int eap_sim_db_resynchronize (void ∗priv, const u8 ∗identity, size_t identity_len, const u8 ∗auts, const u8 ∗_rand)

  Resynchronize AKA AUTN.

6.58.1 Detailed Description

hostapd / EAP-SIM database/authenticator gateway

Copyright

Copyright (c) 2005-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

This is an example implementation of the EAP-SIM/AKA database/authentication gateway interface that is using an external program as an SS7 gateway to GSM/UMTS authentication center (HLR/AuC). hlr_-auc_gw is an example implementation of such a gateway program. This eap_sim_db.c takes care of EAP-SIM/AKA pseudonyms and re-auth identities. It can be used with different gateway implementations for HLR/AuC access. Alternatively, it can also be completely replaced if the in-memory database of pseudonyms/re-auth identities is not suitable for some cases.

Definition in file eap_sim_db.c.

6.58.2 Function Documentation

6.58.2.1 int eap_sim_db_add_pseudonym (void * *priv*, const u8 * *identity*, size_t *identity_len*, char * *pseudonym*)

EAP-SIM DB: Add new pseudonym.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity Identity of the user (may be permanent identity or pseudonym)

identity_len Length of identity

pseudonym Pseudonym for this user. This needs to be an allocated buffer, e.g., return value from eap_sim_db_get_next_pseudonym(). Caller must not free it.

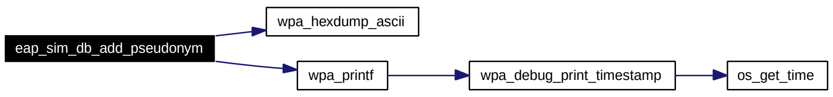
Returns:

0 on success, -1 on failure

This function adds a new pseudonym for EAP-SIM user. EAP-SIM DB is responsible of freeing pseudonym buffer once it is not needed anymore.

Definition at line 902 of file eap_sim_db.c.

Here is the call graph for this function:



6.58.2.2 int eap_sim_db_add_reauth (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, char ∗ *reauth_id*, u16 *counter*, const u8 ∗ *mk*)

EAP-SIM DB: Add new re-authentication entry.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity Identity of the user (may be permanent identity or pseudonym)

identity_len Length of identity

reauth_id reauth_id for this user. This needs to be an allocated buffer, e.g., return value from eap_sim_db_get_next_reauth_id(). Caller must not free it.

mk 16-byte MK from the previous full authentication

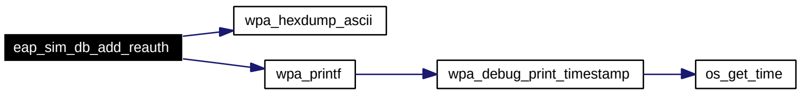
Returns:

0 on success, -1 on failure

This function adds a new re-authentication entry for an EAP-SIM user. EAP-SIM DB is responsible of freeing reauth_id buffer once it is not needed anymore.

Definition at line 963 of file eap_sim_db.c.

Here is the call graph for this function:



6.58.2.3 void eap_sim_db_deinit (void ∗ *priv*)

Deinitialize EAP-SIM DB/authentication gw interface.

Parameters:

priv Private data pointer from eap_sim_db_init()

Definition at line 451 of file eap_sim_db.c.

6.58.2.4 int eap_sim_db_get_aka_auth (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, u8 ∗ *_rand*, u8 ∗ *autn*, u8 ∗ *ik*, u8 ∗ *ck*, u8 ∗ *res*, size_t ∗ *res_len*, void ∗ *cb_session_ctx*)

Get AKA authentication values.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity User name identity

identity_len Length of identity in bytes

_rand Buffer for RAND value

autn Buffer for AUTN value

ik Buffer for IK value

ck Buffer for CK value

res Buffer for RES value

res_len Buffer for RES length

cb_session_ctx Session callback context for get_complete_cb()

Returns:

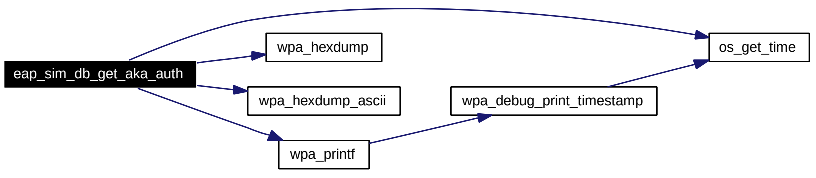
0 on success, -1 (EAP_SIM_DB_FAILURE) on error (e.g., user not found), or -2 (EAP_SIM_DB_-PENDING) if results are not yet available. In this case, the callback function registered with eap_-sim_db_init() will be called once the results become available.

In most cases, the user name is '0' | IMSI, i.e., 0 followed by the IMSI in ASCII format.

When using an external server for AKA authentication, this function can always start a request and return EAP_SIM_DB_PENDING immediately if authentication triplets are not available. Once the authentication data are received, callback function registered with eap_sim_db_init() is called to notify EAP state machine to reprocess the message. This eap_sim_db_get_aka_auth() function will then be called again and the newly received triplets will then be given to the caller.

Definition at line 1121 of file eap_sim_db.c.

Here is the call graph for this function:



6.58.2.5 int eap_sim_db_get_gsm_triplets (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, int *max_chal*, u8 ∗ *_rand*, u8 ∗ *kc*, u8 ∗ *sres*, void ∗ *cb_session_ctx*)

Get GSM triplets.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity User name identity

identity_len Length of identity in bytes

max_chal Maximum number of triplets

_rand Buffer for RAND values

kc Buffer for Kc values

sres Buffer for SRES values

cb_session_ctx Session callback context for get_complete_cb()

Returns:

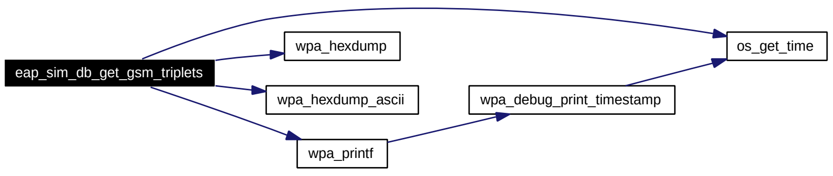
Number of triplets received (has to be less than or equal to max_chal), -1 (EAP_SIM_DB_FAILURE) on error (e.g., user not found), or -2 (EAP_SIM_DB_PENDING) if results are not yet available. In this case, the callback function registered with eap_sim_db_init() will be called once the results become available.

In most cases, the user name is '1' | IMSI, i.e., 1 followed by the IMSI in ASCII format.

When using an external server for GSM triplets, this function can always start a request and return EAP_-SIM_DB_PENDING immediately if authentication triplets are not available. Once the triplets are received, callback function registered with eap_sim_db_init() is called to notify EAP state machine to reprocess the message. This eap_sim_db_get_gsm_triplets() function will then be called again and the newly received triplets will then be given to the caller.

Definition at line 550 of file eap_sim_db.c.

Here is the call graph for this function:



6.58.2.6 char∗ eap_sim_db_get_next_pseudonym (void ∗ *priv*, int *aka*)

EAP-SIM DB: Get next pseudonym.

Parameters:

priv Private data pointer from eap_sim_db_init()

aka Using EAP-AKA instead of EAP-SIM

Returns:

Next pseudonym (allocated string) or NULL on failure

This function is used to generate a pseudonym for EAP-SIM. The returned pseudonym is not added to database at this point; it will need to be added with eap_sim_db_add_pseudonym() once the authentication has been completed successfully. Caller is responsible for freeing the returned buffer.

Definition at line 859 of file eap_sim_db.c.

6.58.2.7 char∗ eap_sim_db_get_next_reauth_id (void ∗ *priv*, int *aka*)

EAP-SIM DB: Get next reauth_id.

Parameters:

priv Private data pointer from eap_sim_db_init()

aka Using EAP-AKA instead of EAP-SIM

Returns:

Next reauth_id (allocated string) or NULL on failure

This function is used to generate a fast re-authentication identity for EAP-SIM. The returned reauth_id is not added to database at this point; it will need to be added with eap_sim_db_add_reauth() once the authentication has been completed successfully. Caller is responsible for freeing the returned buffer.

Definition at line 880 of file eap_sim_db.c.

6.58.2.8 const u8∗ eap_sim_db_get_permanent (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, size_t ∗ *len*)

EAP-SIM DB: Get permanent identity.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity Identity of the user (may be permanent identity or pseudonym)

identity_len Length of identity

len Buffer for length of the returned permanent identity

Returns:

Pointer to the permanent identity, or NULL if not found

Definition at line 1019 of file eap_sim_db.c.

6.58.2.9 struct eap_sim_reauth∗ eap_sim_db_get_reauth_entry (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*)

EAP-SIM DB: Get re-authentication entry.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity Identity of the user (may be permanent identity, pseudonym, or reauth_id)

identity_len Length of identity

len Buffer for length of the returned permanent identity

Returns:

Pointer to the re-auth entry, or NULL if not found

Definition at line 1050 of file eap_sim_db.c.

6.58.2.10 int eap_sim_db_identity_known (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*)

Verify whether the given identity is known.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity User name identity

identity_len Length of identity in bytes

Returns:

0 if the user is found or -1 on failure

In most cases, the user name is ['0','1'] | IMSI, i.e., 1 followed by the IMSI in ASCII format, ['2','3'] | pseudonym, or ['4','5'] | reauth_id.

Definition at line 787 of file eap_sim_db.c.

6.58.2.11 void∗ eap_sim_db_init (const char ∗ *config*, void(∗)(void ∗ctx, void ∗session_ctx) *get_complete_cb*, void ∗ *ctx*)

Initialize EAP-SIM DB / authentication gateway interface.

Parameters:

config Configuration data (e.g., file name)

get_complete_cb Callback function for reporting availability of triplets

ctx Context pointer for get_complete_cb

Returns:

Pointer to a private data structure or NULL on failure

Definition at line 398 of file eap_sim_db.c.

6.58.2.12 void eap_sim_db_remove_reauth (void ∗ *priv*, struct eap_sim_reauth ∗ *reauth*)

EAP-SIM DB: Remove re-authentication entry.

Parameters:

priv Private data pointer from eap_sim_db_init()

reauth Pointer to re-authentication entry from eap_sim_db_get_reauth_entry()

Definition at line 1072 of file eap_sim_db.c.

6.58.2.13 int eap_sim_db_resynchronize (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, const u8 ∗ *auts*, const u8 ∗ *_rand*)

Resynchronize AKA AUTN.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity User name identity

identity_len Length of identity in bytes

auts AUTS value from the peer

_rand RAND value used in the rejected message

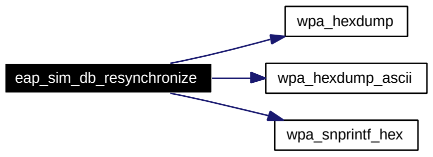
Returns:

0 on success, -1 on failure

This function is called when the peer reports synchronization failure in the AUTN value by sending AUTS. The AUTS and RAND values should be sent to HLR/AuC to allow it to resynchronize with the peer. After this, eap_sim_db_get_aka_auth() will be called again to to fetch updated RAND/AUTN values for the next challenge.

Definition at line 1225 of file eap_sim_db.c.

Here is the call graph for this function:

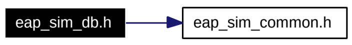

wpa_hexdump
eap_sim_db_resynchronize
wpa_hexdump_ascii
wpa_snprintf_hex

6.59 eap_sim_db.h File Reference

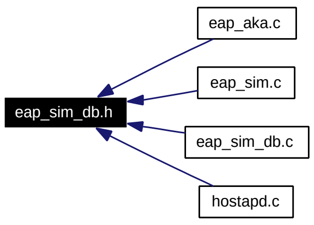
hostapd / EAP-SIM database/authenticator gateway

```
#include "eap_sim_common.h"
```

Include dependency graph for eap_sim_db.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **EAP_SIM_PERMANENT_PREFIX** '1'
- #define **EAP_SIM_PSEUDONYM_PREFIX** '3'
- #define **EAP_SIM_REAUTH_ID_PREFIX** '5'
- #define **EAP_AKA_PERMANENT_PREFIX** '0'
- #define **EAP_AKA_PSEUDONYM_PREFIX** '2'
- #define **EAP_AKA_REAUTH_ID_PREFIX** '4'
- #define **EAP_SIM_DB_FAILURE** -1
- #define **EAP_SIM_DB_PENDING** -2

Functions

- void ∗ eap_sim_db_init (const char ∗config, void(∗get_complete_cb)(void ∗ctx, void ∗session_ctx), void ∗ctx)

  Initialize EAP-SIM DB / authentication gateway interface.

- void eap_sim_db_deinit (void ∗priv)

  Deinitialize EAP-SIM DB/authentication gw interface.

- int eap_sim_db_get_gsm_triplets (void ∗priv, const u8 ∗identity, size_t identity_len, int max_chal, u8 ∗_rand, u8 ∗kc, u8 ∗sres, void ∗cb_session_ctx)

  Get GSM triplets.

- int eap_sim_db_identity_known (void ∗priv, const u8 ∗identity, size_t identity_len)

  Verify whether the given identity is known.

- char ∗ eap_sim_db_get_next_pseudonym (void ∗priv, int aka)

EAP-SIM DB: Get next pseudonym.

- char ∗ eap_sim_db_get_next_reauth_id (void ∗priv, int aka)

  EAP-SIM DB: Get next reauth_id.

- int eap_sim_db_add_pseudonym (void ∗priv, const u8 ∗identity, size_t identity_len, char ∗pseudonym)

  EAP-SIM DB: Add new pseudonym.

- int eap_sim_db_add_reauth (void ∗priv, const u8 ∗identity, size_t identity_len, char ∗reauth_id, u16 counter, const u8 ∗mk)

  EAP-SIM DB: Add new re-authentication entry.

- const u8 ∗ eap_sim_db_get_permanent (void ∗priv, const u8 ∗identity, size_t identity_len, size_t ∗len)

  EAP-SIM DB: Get permanent identity.

- eap_sim_reauth ∗ eap_sim_db_get_reauth_entry (void ∗priv, const u8 ∗identity, size_t identity_-len)

  EAP-SIM DB: Get re-authentication entry.

- void eap_sim_db_remove_reauth (void ∗priv, struct eap_sim_reauth ∗reauth)

  EAP-SIM DB: Remove re-authentication entry.

- int eap_sim_db_get_aka_auth (void ∗priv, const u8 ∗identity, size_t identity_len, u8 ∗_rand, u8 ∗autn, u8 ∗ik, u8 ∗ck, u8 ∗res, size_t ∗res_len, void ∗cb_session_ctx)

  Get AKA authentication values.

- int eap_sim_db_resynchronize (void ∗priv, const u8 ∗identity, size_t identity_len, const u8 ∗auts, const u8 ∗_rand)

  Resynchronize AKA AUTN.

6.59.1 Detailed Description

hostapd / EAP-SIM database/authenticator gateway

Copyright

Definition in file eap_sim_db.h.

6.59.2 Function Documentation

6.59.2.1 int eap_sim_db_add_pseudonym (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, char ∗ *pseudonym*)

EAP-SIM DB: Add new pseudonym.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity Identity of the user (may be permanent identity or pseudonym)

identity_len Length of identity

pseudonym Pseudonym for this user. This needs to be an allocated buffer, e.g., return value from eap_sim_db_get_next_pseudonym(). Caller must not free it.

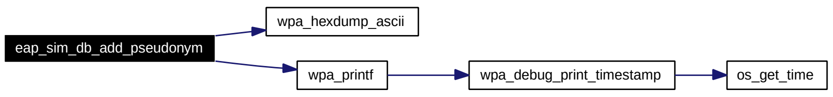
Returns:

0 on success, -1 on failure

This function adds a new pseudonym for EAP-SIM user. EAP-SIM DB is responsible of freeing pseudonym buffer once it is not needed anymore.

Definition at line 902 of file eap_sim_db.c.

Here is the call graph for this function:



6.59.2.2 int eap_sim_db_add_reauth (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, char ∗ *reauth_id*, u16 *counter*, const u8 ∗ *mk*)

EAP-SIM DB: Add new re-authentication entry.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity Identity of the user (may be permanent identity or pseudonym)

identity_len Length of identity

reauth_id reauth_id for this user. This needs to be an allocated buffer, e.g., return value from eap_-sim_db_get_next_reauth_id(). Caller must not free it.

mk 16-byte MK from the previous full authentication

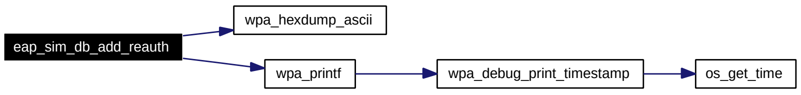
Returns:

0 on success, -1 on failure

This function adds a new re-authentication entry for an EAP-SIM user. EAP-SIM DB is responsible of freeing reauth_id buffer once it is not needed anymore.

Definition at line 963 of file eap_sim_db.c.

Here is the call graph for this function:



6.59.2.3 void eap_sim_db_deinit (void ∗ *priv*)

Deinitialize EAP-SIM DB/authentication gw interface.

Parameters:

priv Private data pointer from eap_sim_db_init()

Definition at line 451 of file eap_sim_db.c.

6.59.2.4 int eap_sim_db_get_aka_auth (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, u8 ∗ *_rand*, u8 ∗ *autn*, u8 ∗ *ik*, u8 ∗ *ck*, u8 ∗ *res*, size_t ∗ *res_len*, void ∗ *cb_session_ctx*)

Get AKA authentication values.

Parameters:

priv	Private data pointer from eap_sim_db_init()
identity	User name identity
identity_len	Length of identity in bytes
_rand	Buffer for RAND value
autn	Buffer for AUTN value
ik	Buffer for IK value
ck	Buffer for CK value
res	Buffer for RES value
res_len	Buffer for RES length
cb_session_ctx	Session callback context for get_complete_cb()

Returns:

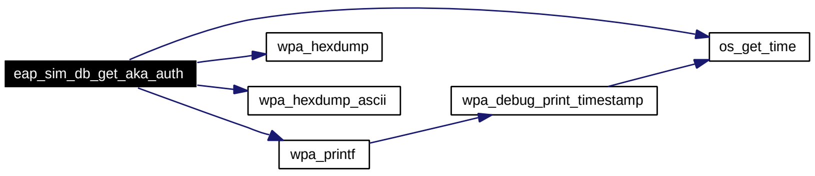
0 on success, -1 (EAP_SIM_DB_FAILURE) on error (e.g., user not found), or -2 (EAP_SIM_DB_-PENDING) if results are not yet available. In this case, the callback function registered with eap_-sim_db_init() will be called once the results become available.

In most cases, the user name is ’0’ | IMSI, i.e., 0 followed by the IMSI in ASCII format.

When using an external server for AKA authentication, this function can always start a request and return EAP_SIM_DB_PENDING immediately if authentication triplets are not available. Once the authentication data are received, callback function registered with eap_sim_db_init() is called to notify EAP state machine to reprocess the message. This eap_sim_db_get_aka_auth() function will then be called again and the newly received triplets will then be given to the caller.

Definition at line 1121 of file eap_sim_db.c.

Here is the call graph for this function:



6.59.2.5 int eap_sim_db_get_gsm_triplets (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, int *max_chal*, u8 ∗ *_rand*, u8 ∗ *kc*, u8 ∗ *sres*, void ∗ *cb_session_ctx*)

Get GSM triplets.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity User name identity

identity_len Length of identity in bytes

max_chal Maximum number of triplets

_rand Buffer for RAND values

kc Buffer for Kc values

sres Buffer for SRES values

cb_session_ctx Session callback context for get_complete_cb()

Returns:

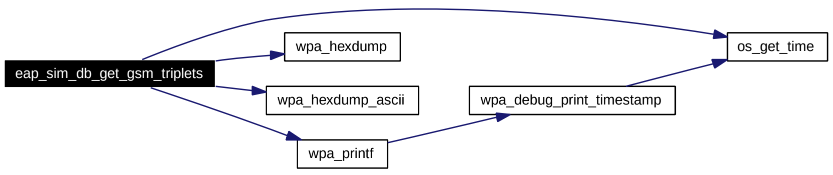
Number of triplets received (has to be less than or equal to max_chal), -1 (EAP_SIM_DB_FAILURE) on error (e.g., user not found), or -2 (EAP_SIM_DB_PENDING) if results are not yet available. In this case, the callback function registered with eap_sim_db_init() will be called once the results become available.

In most cases, the user name is ’1’ | IMSI, i.e., 1 followed by the IMSI in ASCII format.

When using an external server for GSM triplets, this function can always start a request and return EAP_-SIM_DB_PENDING immediately if authentication triplets are not available. Once the triplets are received, callback function registered with eap_sim_db_init() is called to notify EAP state machine to reprocess the message. This eap_sim_db_get_gsm_triplets() function will then be called again and the newly received triplets will then be given to the caller.

Definition at line 550 of file eap_sim_db.c.

Here is the call graph for this function:



6.59.2.6 char∗ eap_sim_db_get_next_pseudonym (void ∗ *priv*, int *aka*)

EAP-SIM DB: Get next pseudonym.

Parameters:

priv Private data pointer from eap_sim_db_init()

aka Using EAP-AKA instead of EAP-SIM

Returns:

Next pseudonym (allocated string) or NULL on failure

This function is used to generate a pseudonym for EAP-SIM. The returned pseudonym is not added to database at this point; it will need to be added with eap_sim_db_add_pseudonym() once the authentication has been completed successfully. Caller is responsible for freeing the returned buffer.

Definition at line 859 of file eap_sim_db.c.

6.59.2.7 char∗ eap_sim_db_get_next_reauth_id (void ∗ *priv*, int *aka*)

EAP-SIM DB: Get next reauth_id.

Parameters:

priv Private data pointer from eap_sim_db_init()

aka Using EAP-AKA instead of EAP-SIM

Returns:

Next reauth_id (allocated string) or NULL on failure

This function is used to generate a fast re-authentication identity for EAP-SIM. The returned reauth_id is not added to database at this point; it will need to be added with eap_sim_db_add_reauth() once the authentication has been completed successfully. Caller is responsible for freeing the returned buffer.

Definition at line 880 of file eap_sim_db.c.

6.59.2.8 const u8∗ eap_sim_db_get_permanent (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, size_t ∗ *len*)

EAP-SIM DB: Get permanent identity.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity Identity of the user (may be permanent identity or pseudonym)

identity_len Length of identity

len Buffer for length of the returned permanent identity

Returns:

Pointer to the permanent identity, or NULL if not found

Definition at line 1019 of file eap_sim_db.c.

6.59.2.9 struct eap_sim_reauth∗ eap_sim_db_get_reauth_entry (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*)

EAP-SIM DB: Get re-authentication entry.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity Identity of the user (may be permanent identity, pseudonym, or reauth_id)

identity_len Length of identity

len Buffer for length of the returned permanent identity

Returns:

Pointer to the re-auth entry, or NULL if not found

Definition at line 1050 of file eap_sim_db.c.

6.59.2.10 int eap_sim_db_identity_known (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*)

Verify whether the given identity is known.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity User name identity

identity_len Length of identity in bytes

Returns:

0 if the user is found or -1 on failure

In most cases, the user name is ['0','1'] | IMSI, i.e., 1 followed by the IMSI in ASCII format, ['2','3'] | pseudonym, or ['4','5'] | reauth_id.

Definition at line 787 of file eap_sim_db.c.

6.59.2.11 void∗ eap_sim_db_init (const char ∗ *config*, void(∗)(void ∗ctx, void ∗session_ctx) *get_complete_cb*, void ∗ *ctx*)

Initialize EAP-SIM DB / authentication gateway interface.

Parameters:

config Configuration data (e.g., file name)

get_complete_cb Callback function for reporting availability of triplets

ctx Context pointer for get_complete_cb

Returns:

Pointer to a private data structure or NULL on failure

Definition at line 398 of file eap_sim_db.c.

6.59.2.12 void eap_sim_db_remove_reauth (void ∗ *priv*, struct eap_sim_reauth ∗ *reauth*)

EAP-SIM DB: Remove re-authentication entry.

Parameters:

priv Private data pointer from eap_sim_db_init()

reauth Pointer to re-authentication entry from eap_sim_db_get_reauth_entry()

Definition at line 1072 of file eap_sim_db.c.

6.59.2.13 int eap_sim_db_resynchronize (void ∗ *priv*, const u8 ∗ *identity*, size_t *identity_len*, const u8 ∗ *auts*, const u8 ∗ *_rand*)

Resynchronize AKA AUTN.

Parameters:

priv Private data pointer from eap_sim_db_init()

identity User name identity

identity_len Length of identity in bytes

auts AUTS value from the peer

_rand RAND value used in the rejected message

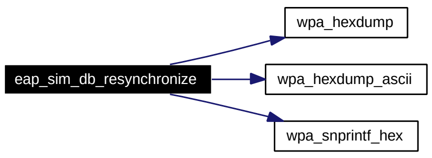
Returns:

0 on success, -1 on failure

This function is called when the peer reports synchronization failure in the AUTN value by sending AUTS. The AUTS and RAND values should be sent to HLR/AuC to allow it to resynchronize with the peer. After this, eap_sim_db_get_aka_auth() will be called again to to fetch updated RAND/AUTN values for the next challenge.

Definition at line 1225 of file eap_sim_db.c.

Here is the call graph for this function:

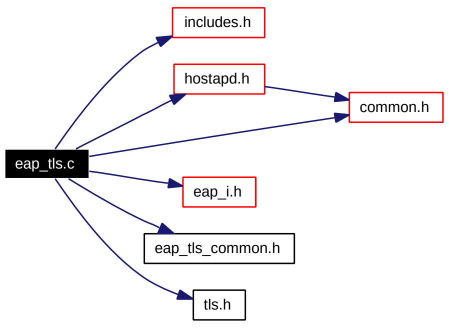


6.60 eap_tls.c File Reference

hostapd / EAP-TLS (RFC 2716)

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
#include "eap_tls_common.h"
#include "tls.h"
```

Include dependency graph for eap_tls.c:



Functions

- int **eap_server_tls_register** (void)

6.60.1 Detailed Description

hostapd / EAP-TLS (RFC 2716)

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

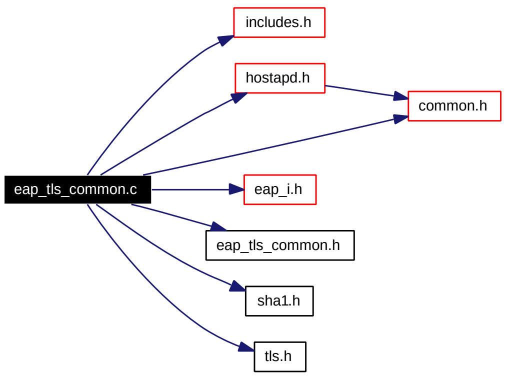
Definition in file eap_tls.c.

6.61 eap_tls_common.c File Reference

hostapd / EAP-TLS/PEAP/TTLS common functions

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
#include "eap_tls_common.h"
#include "sha1.h"
#include "tls.h"
```

Include dependency graph for eap_tls_common.c:



Functions

- int **eap_tls_ssl_init** (struct eap_sm *sm, struct eap_ssl_data *data, int verify_peer)
- void **eap_tls_ssl_deinit** (struct eap_sm *sm, struct eap_ssl_data *data)
- u8 * **eap_tls_derive_key** (struct eap_sm *sm, struct eap_ssl_data *data, char *label, size_t len)
- int **eap_tls_data_reassemble** (struct eap_sm *sm, struct eap_ssl_data *data, u8 **in_data, size_t *in_len)
- int **eap_tls_process_helper** (struct eap_sm *sm, struct eap_ssl_data *data, u8 *in_data, size_t in_len)
- int **eap_tls_buildReq_helper** (struct eap_sm *sm, struct eap_ssl_data *data, int eap_type, int peap_version, u8 id, u8 **out_data, size_t *out_len)
- u8 * **eap_tls_build_ack** (size_t *reqDataLen, u8 id, int eap_type, int peap_version)

6.61.1 Detailed Description

hostapd / EAP-TLS/PEAP/TTLS common functions

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

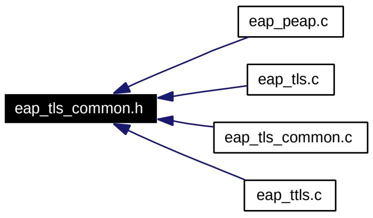
See README and COPYING for more details.

Definition in file eap_tls_common.c.

6.62 eap_tls_common.h File Reference

hostapd / EAP-TLS/PEAP/TTLS common functions

This graph shows which files directly or indirectly include this file:



Defines

- #define **EAP_TLS_FLAGS_LENGTH_INCLUDED** 0x80
- #define **EAP_TLS_FLAGS_MORE_FRAGMENTS** 0x40
- #define **EAP_TLS_FLAGS_START** 0x20
- #define **EAP_PEAP_VERSION_MASK** 0x07
- #define **EAP_TLS_KEY_LEN** 64

Functions

- int **eap_tls_ssl_init** (struct eap_sm ∗sm, struct eap_ssl_data ∗data, int verify_peer)
- void **eap_tls_ssl_deinit** (struct eap_sm ∗sm, struct eap_ssl_data ∗data)
- u8 ∗ **eap_tls_derive_key** (struct eap_sm ∗sm, struct eap_ssl_data ∗data, char ∗label, size_t len)
- int **eap_tls_data_reassemble** (struct eap_sm ∗sm, struct eap_ssl_data ∗data, u8 ∗∗in_data, size_t ∗in_len)
- int **eap_tls_process_helper** (struct eap_sm ∗sm, struct eap_ssl_data ∗data, u8 ∗in_data, size_t in_len)
- int **eap_tls_buildReq_helper** (struct eap_sm ∗sm, struct eap_ssl_data ∗data, int eap_type, int peap_version, u8 id, u8 ∗∗out_data, size_t ∗out_len)
- u8 ∗ **eap_tls_build_ack** (size_t ∗reqDataLen, u8 id, int eap_type, int peap_version)

6.62.1 Detailed Description

hostapd / EAP-TLS/PEAP/TTLS common functions

Copyright

Copyright (c) 2004-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eap_tls_common.h.

6.63 eap_tlv.c File Reference

hostapd / EAP-TLV (draft-josefsson-pppext-eap-tls-eap-07.txt)

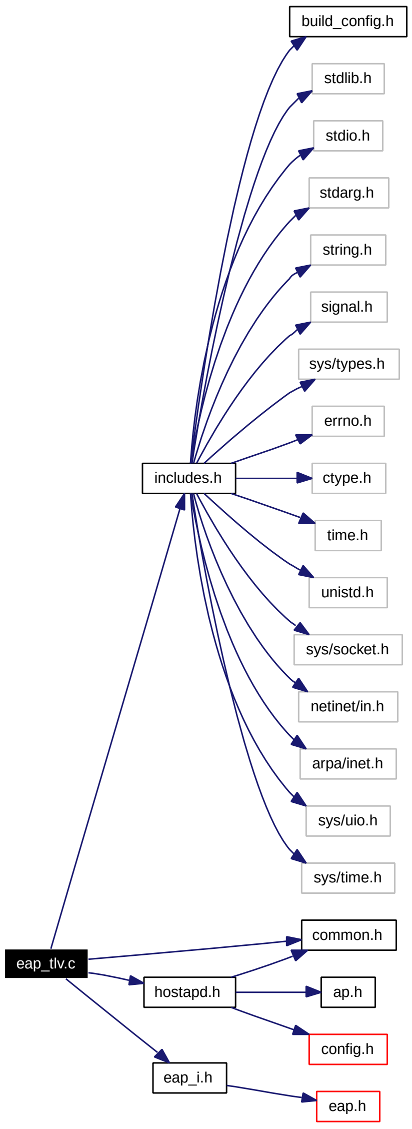
```
#include "includes.h"
```

```
#include "hostapd.h"
```

```
#include "common.h"
```

```
#include "eap_i.h"
```

Include dependency graph for eap_tlv.c:



Defines

- #define **EAP_TLV_RESULT_TLV** 3
- #define **EAP_TLV_NAK_TLV** 4

- #define **EAP_TLV_CRYPTO_BINDING_TLV** 5
- #define **EAP_TLV_CONNECTION_BINDING_TLV** 6
- #define **EAP_TLV_VENDOR_SPECIFIC_TLV** 7
- #define **EAP_TLV_URI_TLV** 8
- #define **EAP_TLV_EAP_PAYLOAD_TLV** 9
- #define **EAP_TLV_INTERMEDIATE_RESULT_TLV** 10
- #define **EAP_TLV_RESULT_SUCCESS** 1
- #define **EAP_TLV_RESULT_FAILURE** 2

Functions

- int **eap_server_tlv_register** (void)

6.63.1 Detailed Description

hostapd / EAP-TLV (draft-josefsson-pppext-eap-tls-eap-07.txt)

Copyright

Copyright (c) 2004, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

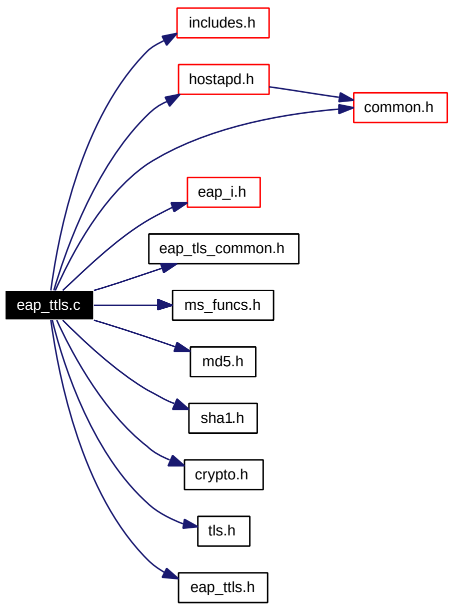
Definition in file eap_tlv.c.

6.64 eap_ttls.c File Reference

hostapd / EAP-TTLS (draft-ietf-pppext-eap-ttls-05.txt)

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "eap_i.h"
#include "eap_tls_common.h"
#include "ms_funcs.h"
#include "md5.h"
#include "sha1.h"
#include "crypto.h"
#include "tls.h"
#include "eap_ttls.h"
```

Include dependency graph for eap_ttls.c:



Defines

- #define **EAP_TTLS_VERSION** 0
- #define **MSCHAPV2_KEY_LEN** 16

Functions

- int **eap_server_ttls_register** (void)

6.64.1 Detailed Description

hostapd / EAP-TTLS (draft-ietf-pppext-eap-ttls-05.txt)

Copyright

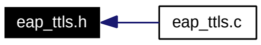
Copyright (c) 2004-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

Definition in file eap_ttls.c.

6.65 eap_ttls.h File Reference

EAP server/peer: EAP-TTLS (draft-ietf-pppext-eap-ttls-03.txt).

This graph shows which files directly or indirectly include this file:



Defines

- #define **AVP_FLAGS_VENDOR** 0x80
- #define **AVP_FLAGS_MANDATORY** 0x40
- #define **AVP_PAD**(start, pos)
- #define **RADIUS_ATTR_USER_NAME** 1
- #define **RADIUS_ATTR_USER_PASSWORD** 2
- #define **RADIUS_ATTR_CHAP_PASSWORD** 3
- #define **RADIUS_ATTR_REPLY_MESSAGE** 18
- #define **RADIUS_ATTR_CHAP_CHALLENGE** 60
- #define **RADIUS_ATTR_EAP_MESSAGE** 79
- #define **RADIUS_VENDOR_ID_MICROSOFT** 311
- #define **RADIUS_ATTR_MS_CHAP_RESPONSE** 1
- #define **RADIUS_ATTR_MS_CHAP_ERROR** 2
- #define **RADIUS_ATTR_MS_CHAP_NT_ENC_PW** 6
- #define **RADIUS_ATTR_MS_CHAP_CHALLENGE** 11
- #define **RADIUS_ATTR_MS_CHAP2_RESPONSE** 25
- #define **RADIUS_ATTR_MS_CHAP2_SUCCESS** 26
- #define **RADIUS_ATTR_MS_CHAP2_CPW** 27
- #define **EAP_TTLS_MSCHAPV2_CHALLENGE_LEN** 16
- #define **EAP_TTLS_MSCHAPV2_RESPONSE_LEN** 50
- #define **EAP_TTLS_MSCHAP_CHALLENGE_LEN** 8
- #define **EAP_TTLS_MSCHAP_RESPONSE_LEN** 50
- #define **EAP_TTLS_CHAP_CHALLENGE_LEN** 16
- #define **EAP_TTLS_CHAP_PASSWORD_LEN** 16

6.65.1 Detailed Description

EAP server/peer: EAP-TTLS (draft-ietf-pppext-eap-ttls-03.txt).

Copyright

Copyright (c) 2004-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eap_ttls.h.

6.65.2 Define Documentation

6.65.2.1 #define AVP_PAD(start, pos)

Value:

```
do { \
        int __pad; \
        __pad = (4 - (((pos) - (start)) & 3)) & 3; \
        os_memset((pos), 0, __pad); \
        pos += __pad; \
} while (0)
```

Definition at line 38 of file eap_ttls.h.

6.66 eap_vendor_test.c File Reference

hostapd / Test method for vendor specific (expanded) EAP type

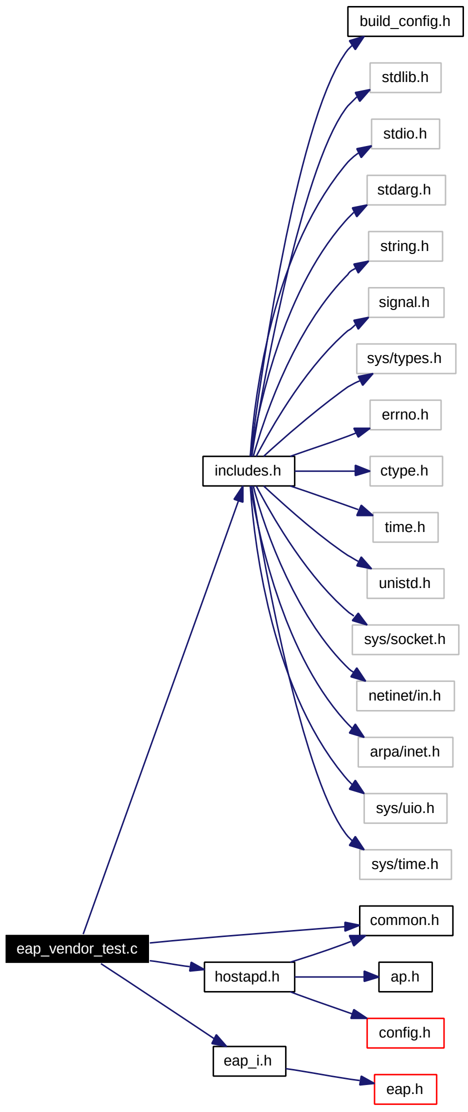
`#include "includes.h"`

`#include "hostapd.h"`

`#include "common.h"`

`#include "eap_i.h"`

Include dependency graph for eap_vendor_test.c:



Defines

- #define **EAP_VENDOR_ID** 0xfffefd
- #define **EAP_VENDOR_TYPE** 0xfcfbfaf9

Functions

- int **eap_server_vendor_test_register** (void)

6.66.1 Detailed Description

hostapd / Test method for vendor specific (expanded) EAP type

Copyright

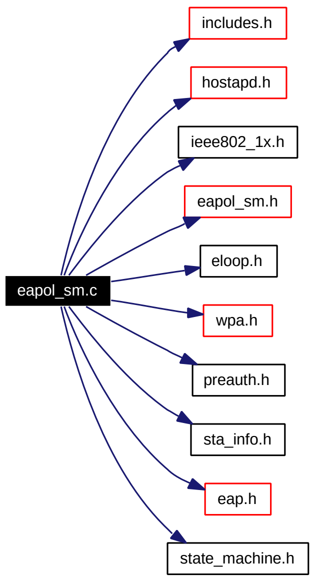
Definition in file eap_vendor_test.c.

6.67 eapol_sm.c File Reference

hostapd / IEEE 802.1X Authenticator - EAPOL state machine

```
#include "includes.h"
#include "hostapd.h"
#include "ieee802_1x.h"
#include "eapol_sm.h"
#include "eloop.h"
#include "wpa.h"
#include "preauth.h"
#include "sta_info.h"
#include "eap.h"
#include "state_machine.h"
```

Include dependency graph for eapol_sm.c:



Defines

- #define **STATE_MACHINE_DATA** struct eapol_state_machine
- #define **STATE_MACHINE_DEBUG_PREFIX** "IEEE 802.1X"
- #define **STATE_MACHINE_ADDR** sm → addr
- #define **setPortAuthorized**() ieee802_1x_set_sta_authorized(sm → hapd, sm → sta, 1)
- #define **setPortUnauthorized**() ieee802_1x_set_sta_authorized(sm → hapd, sm → sta, 0)
- #define **txCannedFail**() ieee802_1x_tx_canned_eap(sm → hapd, sm → sta, 0)
- #define **txCannedSuccess**() ieee802_1x_tx_canned_eap(sm → hapd, sm → sta, 1)
- #define **txReq**() ieee802_1x_tx_req(sm → hapd, sm → sta)
- #define **sendRespToServer**() ieee802_1x_send_resp_to_server(sm → hapd, sm → sta)
- #define **abortAuth**() ieee802_1x_abort_auth(sm → hapd, sm → sta)

- #define **txKey**() ieee802_1x_tx_key(sm → hapd, sm → sta)
- #define **processKey**() do { } while (0)

Functions

- **SM_STATE** (AUTH_PAE, INITIALIZE)
- **SM_STATE** (AUTH_PAE, DISCONNECTED)
- **SM_STATE** (AUTH_PAE, RESTART)
- **SM_STATE** (AUTH_PAE, CONNECTING)
- **SM_STATE** (AUTH_PAE, HELD)
- **SM_STATE** (AUTH_PAE, AUTHENTICATED)
- **SM_STATE** (AUTH_PAE, AUTHENTICATING)
- **SM_STATE** (AUTH_PAE, ABORTING)
- **SM_STATE** (AUTH_PAE, FORCE_AUTH)
- **SM_STATE** (AUTH_PAE, FORCE_UNAUTH)
- **SM_STEP** (AUTH_PAE)
- **SM_STATE** (BE_AUTH, INITIALIZE)
- **SM_STATE** (BE_AUTH, REQUEST)
- **SM_STATE** (BE_AUTH, RESPONSE)
- **SM_STATE** (BE_AUTH, SUCCESS)
- **SM_STATE** (BE_AUTH, FAIL)
- **SM_STATE** (BE_AUTH, TIMEOUT)
- **SM_STATE** (BE_AUTH, IDLE)
- **SM_STATE** (BE_AUTH, IGNORE)
- **SM_STEP** (BE_AUTH)
- **SM_STATE** (REAUTH_TIMER, INITIALIZE)
- **SM_STATE** (REAUTH_TIMER, REAUTHENTICATE)
- **SM_STEP** (REAUTH_TIMER)
- **SM_STATE** (AUTH_KEY_TX, NO_KEY_TRANSMIT)
- **SM_STATE** (AUTH_KEY_TX, KEY_TRANSMIT)
- **SM_STEP** (AUTH_KEY_TX)
- **SM_STATE** (KEY_RX, NO_KEY_RECEIVE)
- **SM_STATE** (KEY_RX, KEY_RECEIVE)
- **SM_STEP** (KEY_RX)
- **SM_STATE** (CTRL_DIR, FORCE_BOTH)
- **SM_STATE** (CTRL_DIR, IN_OR_BOTH)
- **SM_STEP** (CTRL_DIR)
- eapol_state_machine ∗ **eapol_sm_alloc** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **eapol_sm_free** (struct eapol_state_machine ∗sm)
- void **eapol_sm_step** (struct eapol_state_machine ∗sm)
- void **eapol_sm_initialize** (struct eapol_state_machine ∗sm)
- int **eapol_sm_eap_pending_cb** (struct eapol_state_machine ∗sm, void ∗ctx)

6.67.1 Detailed Description

hostapd / IEEE 802.1X Authenticator - EAPOL state machine

Copyright

Copyright (c) 2002-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

Definition in file eapol_sm.c.

6.68 eapol_sm.h File Reference

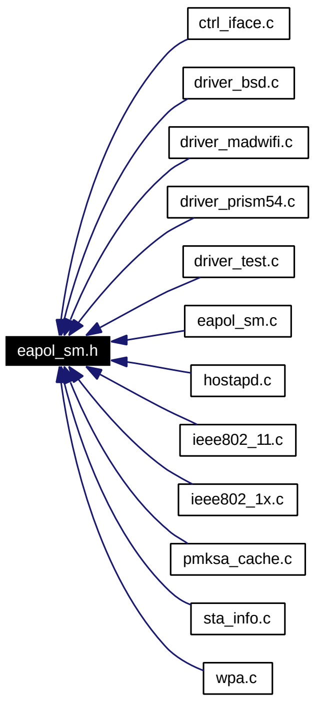
hostapd / IEEE 802.1X Authenticator - EAPOL state machine

`#include "defs.h"`

Include dependency graph for eapol_sm.h:

eapol_sm.h
defs.h

This graph shows which files directly or indirectly include this file:



Defines

- #define **AUTH_PAE_DEFAULT_quietPeriod** 60
- #define **AUTH_PAE_DEFAULT_reAuthMax** 2
- #define **BE_AUTH_DEFAULT_serverTimeout** 30
- #define **EAPOL_SM_PREAUTH** BIT(0)

Typedefs

- typedef unsigned int **Counter**

Enumerations

- enum **PortTypes** { **ForceUnauthorized** = 1, **ForceAuthorized** = 3, **Auto** = 2 }

- enum **PortState** { **Unauthorized** = 2, **Authorized** = 1 }
- enum **ControlledDirection** { **Both** = 0, **In** = 1 }

Functions

- eapol_state_machine ∗ **eapol_sm_alloc** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **eapol_sm_free** (struct eapol_state_machine ∗sm)
- void **eapol_sm_step** (struct eapol_state_machine ∗sm)
- void **eapol_sm_initialize** (struct eapol_state_machine ∗sm)
- void **eapol_sm_dump_state** (FILE ∗f, const char ∗prefix, struct eapol_state_machine ∗sm)
- int **eapol_sm_eap_pending_cb** (struct eapol_state_machine ∗sm, void ∗ctx)

6.68.1 Detailed Description

hostapd / IEEE 802.1X Authenticator - EAPOL state machine

Copyright

Copyright (c) 2002-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

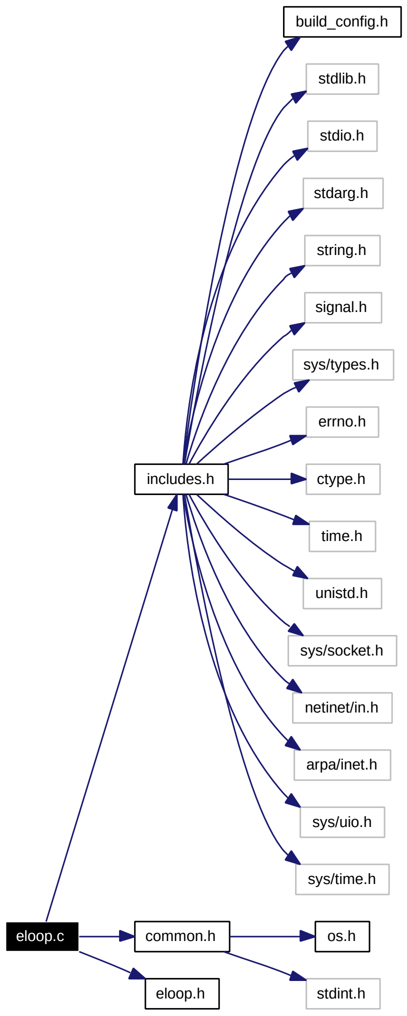
Definition in file eapol_sm.h.

6.69 eloop.c File Reference

Event loop based on select() loop.

```
#include "includes.h"
#include "common.h"
#include "eloop.h"
```

Include dependency graph for eloop.c:



Functions

- int eloop_init (void ∗user_data)

  Initialize global event loop data.

- int eloop_register_read_sock (int sock, eloop_sock_handler handler, void ∗eloop_data, void ∗user_-data)

  Register handler for read events.

- void eloop_unregister_read_sock (int sock)

  Unregister handler for read events.

- int eloop_register_sock (int sock, eloop_event_type type, eloop_sock_handler handler, void ∗eloop_-data, void ∗user_data)

  Register handler for socket events.

- void eloop_unregister_sock (int sock, eloop_event_type type)

  Unregister handler for socket events.

- int eloop_register_timeout (unsigned int secs, unsigned int usecs, eloop_timeout_handler handler, void ∗eloop_data, void ∗user_data)

  Register timeout.

- int eloop_cancel_timeout (eloop_timeout_handler handler, void ∗eloop_data, void ∗user_data)

  Cancel timeouts.

- int eloop_register_signal (int sig, eloop_signal_handler handler, void ∗user_data)

  Register handler for signals.

- int eloop_register_signal_terminate (eloop_signal_handler handler, void ∗user_data)

  Register handler for terminate signals.

- int eloop_register_signal_reconfig (eloop_signal_handler handler, void ∗user_data)

  Register handler for reconfig signals.

- void eloop_run (void)

  Start the event loop.

- void eloop_terminate (void)

  Terminate event loop.

- void eloop_destroy (void)

  Free any resources allocated for the event loop.

- int eloop_terminated (void)

  Check whether event loop has been terminated.

- void eloop_wait_for_read_sock (int sock)

  Wait for a single reader.

- void ∗ eloop_get_user_data (void)

  Get global user data.

6.69.1 Detailed Description

Event loop based on select() loop.

Copyright

Copyright (c) 2002-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eloop.c.

6.69.2 Function Documentation

6.69.2.1 int eloop_cancel_timeout (eloop_timeout_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Cancel timeouts.

Parameters:

handler Matching callback function

eloop_data Matching eloop_data or ELOOP_ALL_CTX to match all

user_data Matching user_data or ELOOP_ALL_CTX to match all

Returns:

Number of cancelled timeouts

Cancel matching <handler,eloop_data,user_data> timeouts registered with eloop_register_timeout(). ELOOP_ALL_CTX can be used as a wildcard for cancelling all timeouts regardless of eloop_data/user_-data.

Definition at line 274 of file eloop.c.

6.69.2.2 void eloop_destroy (void)

Free any resources allocated for the event loop.

After calling eloop_destroy(), other eloop_∗ functions must not be called before re-running eloop_init().

Definition at line 493 of file eloop.c.

6.69.2.3 void∗ eloop_get_user_data (void)

Get global user data.

Returns:

user_data pointer that was registered with eloop_init()

Definition at line 529 of file eloop.c.

6.69.2.4 int eloop_init (void ∗ *user_data*)

Initialize global event loop data.

Parameters:

user_data Pointer to global data passed as eloop_ctx to signal handlers

Returns:

0 on success, -1 on failure

This function must be called before any other eloop_∗ function. user_data can be used to configure a global (to the process) pointer that will be passed as eloop_ctx parameter to signal handlers.

Definition at line 73 of file eloop.c.

6.69.2.5 int eloop_register_read_sock (int *sock*, eloop_sock_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Register handler for read events.

Parameters:

sock File descriptor number for the socket

handler Callback function to be called when data is available for reading

eloop_data Callback context data (eloop_ctx)

user_data Callback context data (sock_ctx)

Returns:

0 on success, -1 on failure

Register a read socket notifier for the given file descriptor. The handler function will be called whenever data is available for reading from the socket. The handler function is responsible for clearing the event after having processed it in order to avoid eloop from calling the handler again for the same event.

Definition at line 177 of file eloop.c.

6.69.2.6 int eloop_register_signal (int *sig*, eloop_signal_handler *handler*, void ∗ *user_data*)

Register handler for signals.

Parameters:

sig Signal number (e.g., SIGHUP)

handler Callback function to be called when the signal is received

user_data Callback context data (signal_ctx)

Returns:

0 on success, -1 on failure

Register a callback function that will be called when a signal is received. The callback function is actually called only after the system signal handler has returned. This means that the normal limits for sighandlers (i.e., only "safe functions" allowed) do not apply for the registered callback.

Signals are 'global' events and there is no local eloop_data pointer like with other handlers. The global user_data pointer registered with eloop_init() will be used as eloop_ctx for signal handlers.

Definition at line 369 of file eloop.c.

6.69.2.7 int eloop_register_signal_reconfig (eloop_signal_handler *handler*, void ∗ *user_data*)

Register handler for reconfig signals.

Parameters:

handler Callback function to be called when the signal is received

user_data Callback context data (signal_ctx)

Returns:

0 on success, -1 on failure

Register a callback function that will be called when a reconfiguration / hangup signal is received. The callback function is actually called only after the system signal handler has returned. This means that the normal limits for sighandlers (i.e., only "safe functions" allowed) do not apply for the registered callback.

Signals are 'global' events and there is no local eloop_data pointer like with other handlers. The global user_data pointer registered with eloop_init() will be used as eloop_ctx for signal handlers.

This function is a more portable version of eloop_register_signal() since the knowledge of exact details of the signals is hidden in eloop implementation. In case of operating systems using signal(), this function registers a handler for SIGHUP.

Definition at line 403 of file eloop.c.

6.69.2.8 int eloop_register_signal_terminate (eloop_signal_handler *handler*, void ∗ *user_data*)

Register handler for terminate signals.

Parameters:

handler Callback function to be called when the signal is received

user_data Callback context data (signal_ctx)

Returns:

0 on success, -1 on failure

Register a callback function that will be called when a process termination signal is received. The callback function is actually called only after the system signal handler has returned. This means that the normal limits for sighandlers (i.e., only "safe functions" allowed) do not apply for the registered callback.

Signals are 'global' events and there is no local eloop_data pointer like with other handlers. The global user_data pointer registered with eloop_init() will be used as eloop_ctx for signal handlers.

This function is a more portable version of eloop_register_signal() since the knowledge of exact details of the signals is hidden in eloop implementation. In case of operating systems using signal(), this function registers handlers for SIGINT and SIGTERM.

Definition at line 393 of file eloop.c.

6.69.2.9 int eloop_register_sock (int *sock*, eloop_event_type *type*, eloop_sock_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Register handler for socket events.

Parameters:

sock File descriptor number for the socket

type Type of event to wait for

handler Callback function to be called when the event is triggered

eloop_data Callback context data (eloop_ctx)

user_data Callback context data (sock_ctx)

Returns:

0 on success, -1 on failure

Register an event notifier for the given socket's file descriptor. The handler function will be called whenever the that event is triggered for the socket. The handler function is responsible for clearing the event after having processed it in order to avoid eloop from calling the handler again for the same event.

Definition at line 206 of file eloop.c.

6.69.2.10 int eloop_register_timeout (unsigned int *secs*, unsigned int *usecs*, eloop_timeout_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Register timeout.

Parameters:

secs Number of seconds to the timeout

usecs Number of microseconds to the timeout

handler Callback function to be called when timeout occurs

eloop_data Callback context data (eloop_ctx)

user_data Callback context data (sock_ctx)

Returns:

0 on success, -1 on failure

Register a timeout that will cause the handler function to be called after given time.

Definition at line 227 of file eloop.c.

6.69.2.11 void eloop_run (void)

Start the event loop.

Start the event loop and continue running as long as there are any registered event handlers. This function is run after event loop has been initialized with event_init() and one or more events have been registered.

Definition at line 414 of file eloop.c.

6.69.2.12 void eloop_terminate (void)

Terminate event loop.

Terminate event loop even if there are registered events. This can be used to request the program to be terminated cleanly.

Definition at line 487 of file eloop.c.

6.69.2.13 int eloop_terminated (void)

Check whether event loop has been terminated.

Returns:

1 = event loop terminate, 0 = event loop still running

This function can be used to check whether eloop_terminate() has been called to request termination of the event loop. This is normally used to abort operations that may still be queued to be run when eloop_-terminate() was called.

Definition at line 510 of file eloop.c.

6.69.2.14 void eloop_unregister_read_sock (int *sock*)

Unregister handler for read events.

Parameters:

sock File descriptor number for the socket

Unregister a read socket notifier that was previously registered with eloop_register_read_sock().

Definition at line 185 of file eloop.c.

6.69.2.15 void eloop_unregister_sock (int *sock*, eloop_event_type *type*)

Unregister handler for socket events.

Parameters:

sock File descriptor number for the socket

type Type of event for which sock was registered

Unregister a socket event notifier that was previously registered with eloop_register_sock().

Definition at line 218 of file eloop.c.

6.69.2.16 void eloop_wait_for_read_sock (int *sock*)

Wait for a single reader.

Parameters:

sock File descriptor number for the socket

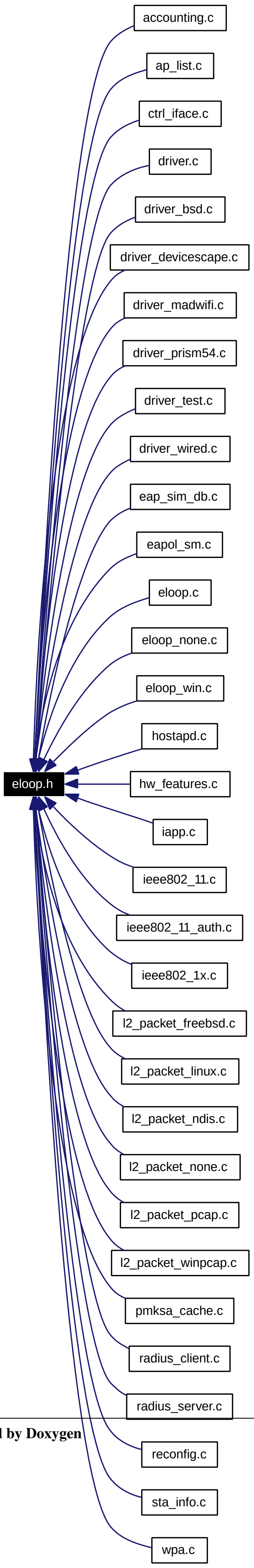
Do a blocking wait for a single read socket.

Definition at line 516 of file eloop.c.

6.70 eloop.h File Reference

Event loop.

This graph shows which files directly or indirectly include this file:


accounting.c
ap_list.c
ctrl_iface.c
driver.c
driver_bsd.c
driver_devicescape.c
driver_madwifi.c
driver_prism54.c
driver_test.c
driver_wired.c
eap_sim_db.c
eapol_sm.c
eloop.c
eloop_none.c
eloop_win.c
hostapd.c
eloop.h
hw_features.c
iapp.c
ieee802_11.c
ieee802_11_auth.c
ieee802_1x.c
l2_packet_freebsd.c
l2_packet_linux.c
l2_packet_ndis.c
l2_packet_none.c
l2_packet_pcap.c
l2_packet_winpcap.c
pmksa_cache.c
radius_client.c
radius_server.c
reconfig.c
sta_info.c
wpa.c

Defines

- #define ELOOP_ALL_CTX (void ∗) -1

  eloop_cancel_timeout() magic number to match all timeouts

Typedefs

- typedef void(∗ eloop_sock_handler)(int sock, void ∗eloop_ctx, void ∗sock_ctx)

  eloop socket event callback type

- typedef void(∗ eloop_event_handler)(void ∗eloop_data, void ∗user_ctx)

  eloop generic event callback type

- typedef void(∗ eloop_timeout_handler)(void ∗eloop_data, void ∗user_ctx)

  eloop timeout event callback type

- typedef void(∗ eloop_signal_handler)(int sig, void ∗eloop_ctx, void ∗signal_ctx)

  eloop signal event callback type

Enumerations

- enum eloop_event_type { **EVENT_TYPE_READ** = 0, **EVENT_TYPE_WRITE**, **EVENT_-TYPE_EXCEPTION** }

  eloop socket event type for eloop_register_sock()

Functions

- int eloop_init (void ∗user_data)

  Initialize global event loop data.

- int eloop_register_read_sock (int sock, eloop_sock_handler handler, void ∗eloop_data, void ∗user_-data)

  Register handler for read events.

- void eloop_unregister_read_sock (int sock)

  Unregister handler for read events.

- int eloop_register_sock (int sock, eloop_event_type type, eloop_sock_handler handler, void ∗eloop_-data, void ∗user_data)

  Register handler for socket events.

- void eloop_unregister_sock (int sock, eloop_event_type type)

  Unregister handler for socket events.

- int eloop_register_event (void ∗event, size_t event_size, eloop_event_handler handler, void ∗eloop_-data, void ∗user_data)

Register handler for generic events.

- void eloop_unregister_event (void ∗event, size_t event_size)

  Unregister handler for a generic event.

- int eloop_register_timeout (unsigned int secs, unsigned int usecs, eloop_timeout_handler handler, void ∗eloop_data, void ∗user_data)

  Register timeout.

- int eloop_cancel_timeout (eloop_timeout_handler handler, void ∗eloop_data, void ∗user_data)

  Cancel timeouts.

- int eloop_register_signal (int sig, eloop_signal_handler handler, void ∗user_data)

  Register handler for signals.

- int eloop_register_signal_terminate (eloop_signal_handler handler, void ∗user_data)

  Register handler for terminate signals.

- int eloop_register_signal_reconfig (eloop_signal_handler handler, void ∗user_data)

  Register handler for reconfig signals.

- void eloop_run (void)

  Start the event loop.

- void eloop_terminate (void)

  Terminate event loop.

- void eloop_destroy (void)

  Free any resources allocated for the event loop.

- int eloop_terminated (void)

  Check whether event loop has been terminated.

- void eloop_wait_for_read_sock (int sock)

  Wait for a single reader.

- void ∗ eloop_get_user_data (void)

  Get global user data.

6.70.1 Detailed Description

Event loop.

Copyright

Copyright (c) 2002-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

This file defines an event loop interface that supports processing events from registered timeouts (i.e., do something after N seconds), sockets (e.g., a new packet available for reading), and signals. eloop.c is an implementation of this interface using select() and sockets. This is suitable for most UNIX/POSIX systems. When porting to other operating systems, it may be necessary to replace that implementation with OS specific mechanisms.

Definition in file eloop.h.

6.70.2 Typedef Documentation

6.70.2.1 typedef void(∗ eloop_event_handler)(void ∗eloop_data, void ∗user_ctx)

eloop generic event callback type

Parameters:

eloop_ctx Registered callback context data (eloop_data)

sock_ctx Registered callback context data (user_data)

Definition at line 61 of file eloop.h.

6.70.2.2 typedef void(∗ eloop_signal_handler)(int sig, void ∗eloop_ctx, void ∗signal_ctx)

eloop signal event callback type

Parameters:

sig Signal number

eloop_ctx Registered callback context data (global user_data from eloop_init() call)

signal_ctx Registered callback context data (user_data from eloop_register_signal(), eloop_register_signal_terminate(), or eloop_register_signal_reconfig() call)

Definition at line 81 of file eloop.h.

6.70.2.3 typedef void(∗ eloop_sock_handler)(int sock, void ∗eloop_ctx, void ∗sock_ctx)

eloop socket event callback type

Parameters:

sock File descriptor number for the socket

eloop_ctx Registered callback context data (eloop_data)

sock_ctx Registered callback context data (user_data)

Definition at line 53 of file eloop.h.

6.70.2.4 typedef void(∗ eloop_timeout_handler)(void ∗eloop_data, void ∗user_ctx)

eloop timeout event callback type

Parameters:

eloop_ctx Registered callback context data (eloop_data)

sock_ctx Registered callback context data (user_data)

Definition at line 69 of file eloop.h.

6.70.3 Enumeration Type Documentation

6.70.3.1 enum eloop_event_type

eloop socket event type for eloop_register_sock()

Parameters:

EVENT_TYPE_READ Socket has data available for reading

EVENT_TYPE_WRITE Socket has room for new data to be written

EVENT_TYPE_EXCEPTION An exception has been reported

Definition at line 40 of file eloop.h.

6.70.4 Function Documentation

6.70.4.1 int eloop_cancel_timeout (eloop_timeout_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Cancel timeouts.

Parameters:

handler Matching callback function

eloop_data Matching eloop_data or ELOOP_ALL_CTX to match all

user_data Matching user_data or ELOOP_ALL_CTX to match all

Returns:

Number of cancelled timeouts

Cancel matching <handler,eloop_data,user_data> timeouts registered with eloop_register_timeout(). ELOOP_ALL_CTX can be used as a wildcard for cancelling all timeouts regardless of eloop_data/user_-data.

Definition at line 274 of file eloop.c.

6.70.4.2 void eloop_destroy (void)

Free any resources allocated for the event loop.

After calling eloop_destroy(), other eloop_∗ functions must not be called before re-running eloop_init().

Definition at line 493 of file eloop.c.

6.70.4.3 void∗ eloop_get_user_data (void)

Get global user data.

Returns:

user_data pointer that was registered with eloop_init()

Definition at line 529 of file eloop.c.

6.70.4.4 int eloop_init (void ∗ *user_data*)

Initialize global event loop data.

Parameters:

user_data Pointer to global data passed as eloop_ctx to signal handlers

Returns:

0 on success, -1 on failure

This function must be called before any other eloop_∗ function. user_data can be used to configure a global (to the process) pointer that will be passed as eloop_ctx parameter to signal handlers.

Definition at line 73 of file eloop.c.

6.70.4.5 int eloop_register_event (void ∗ *event*, size_t *event_size*, eloop_event_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Register handler for generic events.

Parameters:

event Event to wait (eloop implementation specific)

event_size Size of event data

handler Callback function to be called when event is triggered

eloop_data Callback context data (eloop_data)

user_data Callback context data (user_data)

Returns:

0 on success, -1 on failure

Register an event handler for the given event. This function is used to register eloop implementation specific events which are mainly targetted for operating system specific code (driver interface and l2_packet) since the portable code will not be able to use such an OS-specific call. The handler function will be called whenever the event is triggered. The handler function is responsible for clearing the event after having processed it in order to avoid eloop from calling the handler again for the same event.

In case of Windows implementation (eloop_win.c), event pointer is of HANDLE type, i.e., void∗. The callers are likely to have 'HANDLE h' type variable, and they would call this function with eloop_register_-event(h, sizeof(h), ...).

Definition at line 191 of file eloop_win.c.

6.70.4.6 int eloop_register_read_sock (int *sock*, eloop_sock_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Register handler for read events.

Parameters:

sock File descriptor number for the socket

handler Callback function to be called when data is available for reading

eloop_data Callback context data (eloop_ctx)

user_data Callback context data (sock_ctx)

Returns:

0 on success, -1 on failure

Register a read socket notifier for the given file descriptor. The handler function will be called whenever data is available for reading from the socket. The handler function is responsible for clearing the event after having processed it in order to avoid eloop from calling the handler again for the same event.

Definition at line 177 of file eloop.c.

Here is the call graph for this function:

eloop_register_read_sock
eloop_register_sock

6.70.4.7 int eloop_register_signal (int *sig*, eloop_signal_handler *handler*, void ∗ *user_data*)

Register handler for signals.

Parameters:

sig Signal number (e.g., SIGHUP)

handler Callback function to be called when the signal is received

user_data Callback context data (signal_ctx)

Returns:

0 on success, -1 on failure

Register a callback function that will be called when a signal is received. The callback function is actually called only after the system signal handler has returned. This means that the normal limits for sighandlers (i.e., only "safe functions" allowed) do not apply for the registered callback.

Signals are ’global’ events and there is no local eloop_data pointer like with other handlers. The global user_data pointer registered with eloop_init() will be used as eloop_ctx for signal handlers.

Definition at line 369 of file eloop.c.

6.70.4.8 int eloop_register_signal_reconfig (eloop_signal_handler *handler*, void ∗ *user_data*)

Register handler for reconfig signals.

Parameters:

handler Callback function to be called when the signal is received

user_data Callback context data (signal_ctx)

Returns:

0 on success, -1 on failure

Register a callback function that will be called when a reconfiguration / hangup signal is received. The callback function is actually called only after the system signal handler has returned. This means that the normal limits for sighandlers (i.e., only "safe functions" allowed) do not apply for the registered callback.

Signals are 'global' events and there is no local eloop_data pointer like with other handlers. The global user_data pointer registered with eloop_init() will be used as eloop_ctx for signal handlers.

This function is a more portable version of eloop_register_signal() since the knowledge of exact details of the signals is hidden in eloop implementation. In case of operating systems using signal(), this function registers a handler for SIGHUP.

Definition at line 403 of file eloop.c.

Here is the call graph for this function:

eloop_register_signal_reconfig
eloop_register_signal

6.70.4.9 int eloop_register_signal_terminate (eloop_signal_handler *handler*, void ∗ *user_data*)

Register handler for terminate signals.

Parameters:

handler Callback function to be called when the signal is received

user_data Callback context data (signal_ctx)

Returns:

0 on success, -1 on failure

Register a callback function that will be called when a process termination signal is received. The callback function is actually called only after the system signal handler has returned. This means that the normal limits for sighandlers (i.e., only "safe functions" allowed) do not apply for the registered callback.

Signals are 'global' events and there is no local eloop_data pointer like with other handlers. The global user_data pointer registered with eloop_init() will be used as eloop_ctx for signal handlers.

This function is a more portable version of eloop_register_signal() since the knowledge of exact details of the signals is hidden in eloop implementation. In case of operating systems using signal(), this function registers handlers for SIGINT and SIGTERM.

Definition at line 393 of file eloop.c.

Here is the call graph for this function:

eloop_register_signal_terminate
eloop_register_signal

6.70.4.10 int eloop_register_sock (int *sock*, eloop_event_type *type*, eloop_sock_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Register handler for socket events.

Parameters:

sock File descriptor number for the socket

type Type of event to wait for

handler Callback function to be called when the event is triggered

eloop_data Callback context data (eloop_ctx)

user_data Callback context data (sock_ctx)

Returns:

0 on success, -1 on failure

Register an event notifier for the given socket's file descriptor. The handler function will be called whenever the that event is triggered for the socket. The handler function is responsible for clearing the event after having processed it in order to avoid eloop from calling the handler again for the same event.

Definition at line 206 of file eloop.c.

6.70.4.11 int eloop_register_timeout (unsigned int *secs*, unsigned int *usecs*, eloop_timeout_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Register timeout.

Parameters:

secs Number of seconds to the timeout

usecs Number of microseconds to the timeout

handler Callback function to be called when timeout occurs

eloop_data Callback context data (eloop_ctx)

user_data Callback context data (sock_ctx)

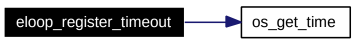
Returns:

0 on success, -1 on failure

Register a timeout that will cause the handler function to be called after given time.

Definition at line 227 of file eloop.c.

Here is the call graph for this function:



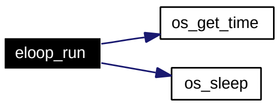
6.70.4.12 void eloop_run (void)

Start the event loop.

Start the event loop and continue running as long as there are any registered event handlers. This function is run after event loop has been initialized with event_init() and one or more events have been registered.

Definition at line 414 of file eloop.c.

Here is the call graph for this function:



6.70.4.13 void eloop_terminate (void)

Terminate event loop.

Terminate event loop even if there are registered events. This can be used to request the program to be terminated cleanly.

Definition at line 487 of file eloop.c.

6.70.4.14 int eloop_terminated (void)

Check whether event loop has been terminated.

Returns:
1 = event loop terminate, 0 = event loop still running

This function can be used to check whether eloop_terminate() has been called to request termination of the event loop. This is normally used to abort operations that may still be queued to be run when eloop_-terminate() was called.

Definition at line 510 of file eloop.c.

6.70.4.15 void eloop_unregister_event (void ∗ *event*, size_t *event_size*)

Unregister handler for a generic event.

Parameters:
event Event to cancel (eloop implementation specific)
event_size Size of event data

Unregister a generic event notifier that was previously registered with eloop_register_event().

Definition at line 220 of file eloop_win.c.

6.70.4.16 void eloop_unregister_read_sock (int *sock*)

Unregister handler for read events.

Parameters:
sock File descriptor number for the socket

Unregister a read socket notifier that was previously registered with eloop_register_read_sock().

Definition at line 185 of file eloop.c.

Here is the call graph for this function:

eloop_unregister_read_sock
eloop_unregister_sock

6.70.4.17 void eloop_unregister_sock (int *sock*, eloop_event_type *type*)

Unregister handler for socket events.

Parameters:

sock File descriptor number for the socket

type Type of event for which sock was registered

Unregister a socket event notifier that was previously registered with eloop_register_sock().

Definition at line 218 of file eloop.c.

6.70.4.18 void eloop_wait_for_read_sock (int *sock*)

Wait for a single reader.

Parameters:

sock File descriptor number for the socket

Do a blocking wait for a single read socket.

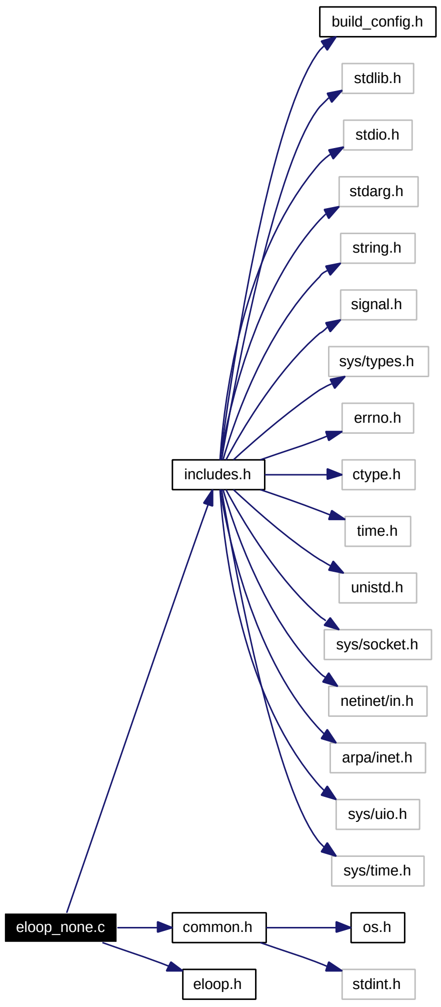
Definition at line 516 of file eloop.c.

6.71 eloop_none.c File Reference

Event loop - empty template (basic structure, but no OS specific operations).

```
#include "includes.h"
#include "common.h"
#include "eloop.h"
```

Include dependency graph for eloop_none.c:



Data Structures

- struct **eloop_sock**
- struct **eloop_timeout**
- struct **eloop_signal**
- struct **eloop_data**

Functions

- int eloop_init (void ∗user_data)

    Initialize global event loop data.

- int **eloop_register_read_sock** (int sock, void(∗handler)(int sock, void ∗eloop_ctx, void ∗sock_ctx), void ∗eloop_data, void ∗user_data)
- void eloop_unregister_read_sock (int sock)

    Unregister handler for read events.

- int **eloop_register_timeout** (unsigned int secs, unsigned int usecs, void(∗handler)(void ∗eloop_ctx, void ∗timeout_ctx), void ∗eloop_data, void ∗user_data)
- int **eloop_cancel_timeout** (void(∗handler)(void ∗eloop_ctx, void ∗sock_ctx), void ∗eloop_data, void ∗user_data)
- int **eloop_register_signal** (int sig, void(∗handler)(int sig, void ∗eloop_ctx, void ∗signal_ctx), void ∗user_data)
- int **eloop_register_signal_terminate** (void(∗handler)(int sig, void ∗eloop_ctx, void ∗signal_ctx), void ∗user_data)
- int **eloop_register_signal_reconfig** (void(∗handler)(int sig, void ∗eloop_ctx, void ∗signal_ctx), void ∗user_data)
- void eloop_run (void)

    Start the event loop.

- void eloop_terminate (void)

    Terminate event loop.

- void eloop_destroy (void)

    Free any resources allocated for the event loop.

- int eloop_terminated (void)

    Check whether event loop has been terminated.

- void eloop_wait_for_read_sock (int sock)

    Wait for a single reader.

- void ∗ eloop_get_user_data (void)

    Get global user data.

6.71.1 Detailed Description

Event loop - empty template (basic structure, but no OS specific operations).

Copyright

Copyright (c) 2002-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file eloop_none.c.

6.71.2 Function Documentation

6.71.2.1 void eloop_destroy (void)

Free any resources allocated for the event loop.

After calling eloop_destroy(), other eloop_∗ functions must not be called before re-running eloop_init().

Definition at line 358 of file eloop_none.c.

6.71.2.2 void∗ eloop_get_user_data (void)

Get global user data.

Returns:
user_data pointer that was registered with eloop_init()

Definition at line 388 of file eloop_none.c.

6.71.2.3 int eloop_init (void ∗ *user_data*)

Initialize global event loop data.

Parameters:
user_data Pointer to global data passed as eloop_ctx to signal handlers

Returns:
0 on success, -1 on failure

This function must be called before any other eloop_∗ function. user_data can be used to configure a global (to the process) pointer that will be passed as eloop_ctx parameter to signal handlers.

Definition at line 64 of file eloop_none.c.

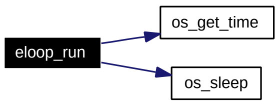
6.71.2.4 void eloop_run (void)

Start the event loop.

Start the event loop and continue running as long as there are any registered event handlers. This function is run after event loop has been initialized with event_init() and one or more events have been registered.

Definition at line 295 of file eloop_none.c.

Here is the call graph for this function:



6.71.2.5 void eloop_terminate (void)

Terminate event loop.

Terminate event loop even if there are registered events. This can be used to request the program to be terminated cleanly.

Definition at line 352 of file eloop_none.c.

6.71.2.6 int eloop_terminated (void)

Check whether event loop has been terminated.

Returns:

1 = event loop terminate, 0 = event loop still running

This function can be used to check whether eloop_terminate() has been called to request termination of the event loop. This is normally used to abort operations that may still be queued to be run when eloop_-terminate() was called.

Definition at line 373 of file eloop_none.c.

6.71.2.7 void eloop_unregister_read_sock (int *sock*)

Unregister handler for read events.

Parameters:

sock File descriptor number for the socket

Unregister a read socket notifier that was previously registered with eloop_register_read_sock().

Definition at line 99 of file eloop_none.c.

6.71.2.8 void eloop_wait_for_read_sock (int *sock*)

Wait for a single reader.

Parameters:

sock File descriptor number for the socket

Do a blocking wait for a single read socket.

Definition at line 379 of file eloop_none.c.

6.72 eloop_win.c File Reference

Event loop based on Windows events and WaitForMultipleObjects.

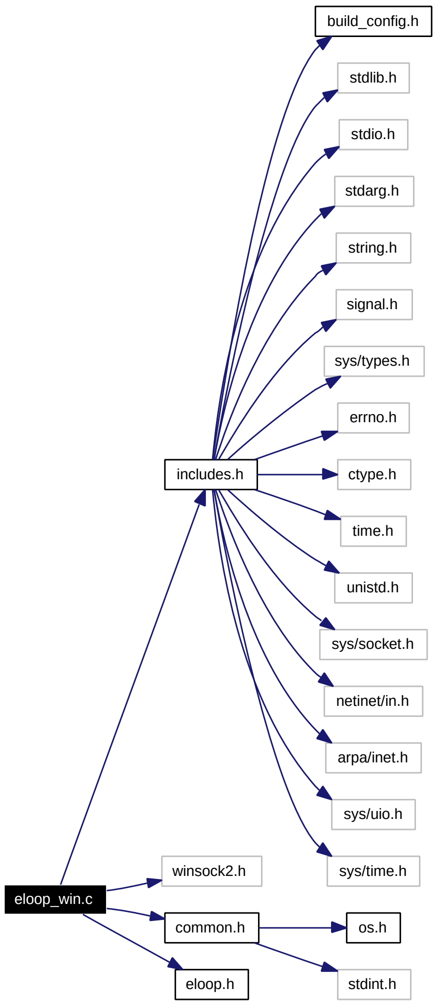
`#include "includes.h"`

`#include <winsock2.h>`

`#include "common.h"`

`#include "eloop.h"`

Include dependency graph for eloop_win.c:



Data Structures

- struct **eloop_sock**
- struct **eloop_timeout**
- struct **eloop_signal**
- struct **eloop_data**

Functions

- int eloop_init (void ∗user_data)

  Initialize global event loop data.

- int eloop_register_read_sock (int sock, eloop_sock_handler handler, void ∗eloop_data, void ∗user_-data)

  Register handler for read events.

- void eloop_unregister_read_sock (int sock)

  Unregister handler for read events.

- int eloop_register_event (void ∗event, size_t event_size, eloop_event_handler handler, void ∗eloop_-data, void ∗user_data)

  Register handler for generic events.

- void eloop_unregister_event (void ∗event, size_t event_size)

  Unregister handler for a generic event.

- int eloop_register_timeout (unsigned int secs, unsigned int usecs, eloop_timeout_handler handler, void ∗eloop_data, void ∗user_data)

  Register timeout.

- int eloop_cancel_timeout (eloop_timeout_handler handler, void ∗eloop_data, void ∗user_data)

  Cancel timeouts.

- int eloop_register_signal (int sig, eloop_signal_handler handler, void ∗user_data)

  Register handler for signals.

- int eloop_register_signal_terminate (eloop_signal_handler handler, void ∗user_data)

  Register handler for terminate signals.

- int eloop_register_signal_reconfig (eloop_signal_handler handler, void ∗user_data)

  Register handler for reconfig signals.

- void eloop_run (void)

  Start the event loop.

- void eloop_terminate (void)

  Terminate event loop.

- void eloop_destroy (void)

  Free any resources allocated for the event loop.

- int eloop_terminated (void)

  Check whether event loop has been terminated.

- void eloop_wait_for_read_sock (int sock)

  Wait for a single reader.

- void ∗ eloop_get_user_data (void)

  Get global user data.

6.72.1 Detailed Description

Event loop based on Windows events and WaitForMultipleObjects.

Copyright

Definition in file eloop_win.c.

6.72.2 Function Documentation

6.72.2.1 int eloop_cancel_timeout (eloop_timeout_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Cancel timeouts.

Parameters:

handler	Matching callback function
eloop_data	Matching eloop_data or ELOOP_ALL_CTX to match all
user_data	Matching user_data or ELOOP_ALL_CTX to match all

Returns:

Number of cancelled timeouts

Cancel matching <handler,eloop_data,user_data> timeouts registered with eloop_register_timeout(). ELOOP_ALL_CTX can be used as a wildcard for cancelling all timeouts regardless of eloop_data/user_-data.

Definition at line 292 of file eloop_win.c.

6.72.2.2 void eloop_destroy (void)

Free any resources allocated for the event loop.

After calling eloop_destroy(), other eloop_∗ functions must not be called before re-running eloop_init().

Definition at line 553 of file eloop_win.c.

6.72.2.3 void∗ eloop_get_user_data (void)

Get global user data.

Returns:

user_data pointer that was registered with eloop_init()

Definition at line 602 of file eloop_win.c.

6.72.2.4 int eloop_init (void ∗ *user_data*)

Initialize global event loop data.

Parameters:

user_data Pointer to global data passed as eloop_ctx to signal handlers

Returns:

0 on success, -1 on failure

This function must be called before any other eloop_∗ function. user_data can be used to configure a global (to the process) pointer that will be passed as eloop_ctx parameter to signal handlers.

Definition at line 83 of file eloop_win.c.

6.72.2.5 int eloop_register_event (void ∗ *event*, size_t *event_size*, eloop_event_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Register handler for generic events.

Parameters:

event Event to wait (eloop implementation specific)

event_size Size of event data

handler Callback function to be called when event is triggered

eloop_data Callback context data (eloop_data)

user_data Callback context data (user_data)

Returns:

0 on success, -1 on failure

Register an event handler for the given event. This function is used to register eloop implementation specific events which are mainly targetted for operating system specific code (driver interface and l2_packet) since the portable code will not be able to use such an OS-specific call. The handler function will be called whenever the event is triggered. The handler function is responsible for clearing the event after having processed it in order to avoid eloop from calling the handler again for the same event.

In case of Windows implementation (eloop_win.c), event pointer is of HANDLE type, i.e., void∗. The callers are likely to have 'HANDLE h' type variable, and they would call this function with eloop_register_-event(h, sizeof(h), ...).

Definition at line 191 of file eloop_win.c.

6.72.2.6 int eloop_register_read_sock (int *sock*, eloop_sock_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Register handler for read events.

Parameters:

sock File descriptor number for the socket

handler Callback function to be called when data is available for reading

eloop_data Callback context data (eloop_ctx)

user_data Callback context data (sock_ctx)

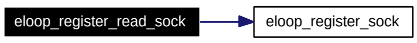
Returns:

0 on success, -1 on failure

Register a read socket notifier for the given file descriptor. The handler function will be called whenever data is available for reading from the socket. The handler function is responsible for clearing the event after having processed it in order to avoid eloop from calling the handler again for the same event.

Definition at line 121 of file eloop_win.c.

Here is the call graph for this function:



6.72.2.7 int eloop_register_signal (int *sig*, eloop_signal_handler *handler*, void ∗ *user_data*)

Register handler for signals.

Parameters:

sig Signal number (e.g., SIGHUP)

handler Callback function to be called when the signal is received

user_data Callback context data (signal_ctx)

Returns:

0 on success, -1 on failure

Register a callback function that will be called when a signal is received. The callback function is actually called only after the system signal handler has returned. This means that the normal limits for sighandlers (i.e., only "safe functions" allowed) do not apply for the registered callback.

Signals are 'global' events and there is no local eloop_data pointer like with other handlers. The global user_data pointer registered with eloop_init() will be used as eloop_ctx for signal handlers.

Definition at line 371 of file eloop_win.c.

6.72.2.8 int eloop_register_signal_reconfig (eloop_signal_handler *handler*, void ∗ *user_data*)

Register handler for reconfig signals.

Parameters:

handler Callback function to be called when the signal is received

user_data Callback context data (signal_ctx)

Returns:

0 on success, -1 on failure

Register a callback function that will be called when a reconfiguration / hangup signal is received. The callback function is actually called only after the system signal handler has returned. This means that the normal limits for sighandlers (i.e., only "safe functions" allowed) do not apply for the registered callback.

Signals are 'global' events and there is no local eloop_data pointer like with other handlers. The global user_data pointer registered with eloop_init() will be used as eloop_ctx for signal handlers.

This function is a more portable version of eloop_register_signal() since the knowledge of exact details of the signals is hidden in eloop implementation. In case of operating systems using signal(), this function registers a handler for SIGHUP.

Definition at line 431 of file eloop_win.c.

Here is the call graph for this function:

eloop_register_signal_reconfig
eloop_register_signal

6.72.2.9 int eloop_register_signal_terminate (eloop_signal_handler *handler*, void ∗ *user_data*)

Register handler for terminate signals.

Parameters:

handler Callback function to be called when the signal is received

user_data Callback context data (signal_ctx)

Returns:

0 on success, -1 on failure

Register a callback function that will be called when a process termination signal is received. The callback function is actually called only after the system signal handler has returned. This means that the normal limits for sighandlers (i.e., only "safe functions" allowed) do not apply for the registered callback.

Signals are 'global' events and there is no local eloop_data pointer like with other handlers. The global user_data pointer registered with eloop_init() will be used as eloop_ctx for signal handlers.

This function is a more portable version of eloop_register_signal() since the knowledge of exact details of the signals is hidden in eloop implementation. In case of operating systems using signal(), this function registers handlers for SIGINT and SIGTERM.

Definition at line 412 of file eloop_win.c.

Here is the call graph for this function:

eloop_register_signal_terminate
eloop_register_signal

6.72.2.10 int eloop_register_timeout (unsigned int *secs*, unsigned int *usecs*, eloop_timeout_handler *handler*, void ∗ *eloop_data*, void ∗ *user_data*)

Register timeout.

Parameters:

secs Number of seconds to the timeout

usecs Number of microseconds to the timeout

handler Callback function to be called when timeout occurs

eloop_data Callback context data (eloop_ctx)

user_data Callback context data (sock_ctx)

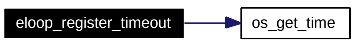
Returns:

0 on success, -1 on failure

Register a timeout that will cause the handler function to be called after given time.

Definition at line 245 of file eloop_win.c.

Here is the call graph for this function:



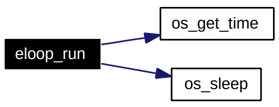
6.72.2.11 void eloop_run (void)

Start the event loop.

Start the event loop and continue running as long as there are any registered event handlers. This function is run after event loop has been initialized with event_init() and one or more events have been registered.

Definition at line 439 of file eloop_win.c.

Here is the call graph for this function:



6.72.2.12 void eloop_terminate (void)

Terminate event loop.

Terminate event loop even if there are registered events. This can be used to request the program to be terminated cleanly.

Definition at line 546 of file eloop_win.c.

6.72.2.13 int eloop_terminated (void)

Check whether event loop has been terminated.

Returns:

1 = event loop terminate, 0 = event loop still running

This function can be used to check whether eloop_terminate() has been called to request termination of the event loop. This is normally used to abort operations that may still be queued to be run when eloop_-terminate() was called.

Definition at line 574 of file eloop_win.c.

6.72.2.14 void eloop_unregister_event (void ∗ *event*, size_t *event_size*)

Unregister handler for a generic event.

Parameters:

event Event to cancel (eloop implementation specific)

event_size Size of event data

Unregister a generic event notifier that was previously registered with eloop_register_event().

Definition at line 220 of file eloop_win.c.

6.72.2.15 void eloop_unregister_read_sock (int *sock*)

Unregister handler for read events.

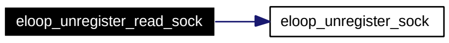
Parameters:

sock File descriptor number for the socket

Unregister a read socket notifier that was previously registered with eloop_register_read_sock().

Definition at line 164 of file eloop_win.c.

Here is the call graph for this function:



6.72.2.16 void eloop_wait_for_read_sock (int *sock*)

Wait for a single reader.

Parameters:

sock File descriptor number for the socket

Do a blocking wait for a single read socket.

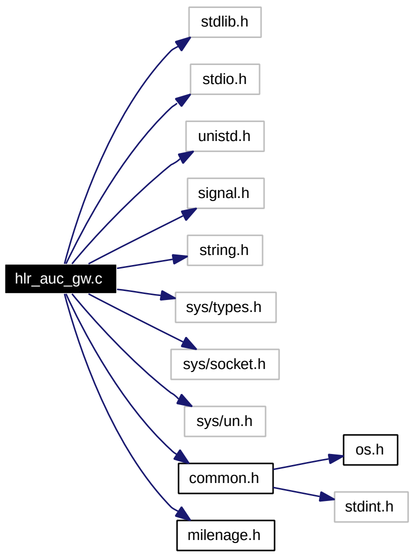
Definition at line 580 of file eloop_win.c.

6.73 hlr_auc_gw.c File Reference

HLR/AuC testing gateway for hostapd EAP-SIM/AKA database/authenticator.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include "common.h"
#include "milenage.h"
```

Include dependency graph for hlr_auc_gw.c:



Defines

- #define **EAP_SIM_MAX_CHAL** 3
- #define **EAP_AKA_RAND_LEN** 16
- #define **EAP_AKA_AUTN_LEN** 16
- #define **EAP_AKA_AUTS_LEN** 14
- #define **EAP_AKA_RES_MAX_LEN** 16
- #define **EAP_AKA_IK_LEN** 16
- #define **EAP_AKA_CK_LEN** 16

Functions

- int **main** (int argc, char ∗argv[ ])

6.73.1 Detailed Description

HLR/AuC testing gateway for hostapd EAP-SIM/AKA database/authenticator.

Copyright

Copyright (c) 2005-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This is an example implementation of the EAP-SIM/AKA database/authentication gateway interface to HLR/AuC. It is expected to be replaced with an implementation of SS7 gateway to GSM/UMTS authentication center (HLR/AuC) or a local implementation of SIM triplet and AKA authentication data generator.

hostapd will send SIM/AKA authentication queries over a UNIX domain socket to and external program, e.g., this hlr_auc_gw. This interface uses simple text-based format:

EAP-SIM / GSM triplet query/response: SIM-REQ-AUTH <imsi> <max_chal> SIM-RESP-AUTH <imsi> Kc1:SRES1:RAND1 Kc2:SRES2:RAND2 [Kc3:SRES3:RAND3] SIM-RESP-AUTH <imsi> FAILURE

EAP-AKA / UMTS query/response: AKA-REQ-AUTH <imsi> AKA-RESP-AUTH <imsi> <rand> <autn> <ik> <ck> <res> AKA-RESP-AUTH <imsi> FAILURE

EAP-AKA / UMTS AUTS (re-synchronization): AKA-AUTS <imsi> <auts> <rand>

IMSI and max_chal are sent as an ASCII string, Kc/SRES/RAND/AUTN/IK/CK/RES/AUTS as hex strings.

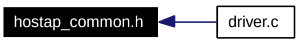
The example implementation here reads GSM authentication triplets from a text file in IMSI:Kc:SRES:RAND format, IMSI in ASCII, other fields as hex strings. This is used to simulate an HLR/AuC. As such, it is not very useful for real life authentication, but it is useful both as an example implementation and for EAP-SIM testing.

Definition in file hlr_auc_gw.c.

6.74 hostap_common.h File Reference

hostapd / Kernel driver communication with Linux Host AP driver

This graph shows which files directly or indirectly include this file:



Defines

- #define **PRISM2_IOCTL_PRISM2_PARAM** (SIOCIWFIRSTPRIV + 0)
- #define **PRISM2_IOCTL_GET_PRISM2_PARAM** (SIOCIWFIRSTPRIV + 1)
- #define **PRISM2_IOCTL_WRITEMIF** (SIOCIWFIRSTPRIV + 2)
- #define **PRISM2_IOCTL_READMIF** (SIOCIWFIRSTPRIV + 3)
- #define **PRISM2_IOCTL_MONITOR** (SIOCIWFIRSTPRIV + 4)
- #define **PRISM2_IOCTL_RESET** (SIOCIWFIRSTPRIV + 6)
- #define **PRISM2_IOCTL_INQUIRE** (SIOCIWFIRSTPRIV + 8)
- #define **PRISM2_IOCTL_WDS_ADD** (SIOCIWFIRSTPRIV + 10)
- #define **PRISM2_IOCTL_WDS_DEL** (SIOCIWFIRSTPRIV + 12)
- #define **PRISM2_IOCTL_SET_RID_WORD** (SIOCIWFIRSTPRIV + 14)
- #define **PRISM2_IOCTL_MACCMD** (SIOCIWFIRSTPRIV + 16)
- #define **PRISM2_IOCTL_ADDMAC** (SIOCIWFIRSTPRIV + 18)
- #define **PRISM2_IOCTL_DELMAC** (SIOCIWFIRSTPRIV + 20)
- #define **PRISM2_IOCTL_KICKMAC** (SIOCIWFIRSTPRIV + 22)
- #define **PRISM2_IOCTL_DOWNLOAD** (SIOCDEVPRIVATE + 13)
- #define **PRISM2_IOCTL_HOSTAPD** (SIOCDEVPRIVATE + 14)
- #define **PRISM2_MAX_DOWNLOAD_AREA_LEN** 131072
- #define **PRISM2_MAX_DOWNLOAD_LEN** 262144
- #define **PRISM2_HOSTAPD_MAX_BUF_SIZE** 1024
- #define **PRISM2_HOSTAPD_RID_HDR_LEN** ((int) (&((struct prism2_hostapd_param ∗) 0) → u.rid.data))
- #define **PRISM2_HOSTAPD_GENERIC_ELEMENT_HDR_LEN** ((int) (&((struct prism2_-hostapd_param ∗) 0) → u.generic_elem.data))
- #define **HOSTAP_CRYPT_ALG_NAME_LEN** 16
- #define **MLME_STA_DEAUTH** 0
- #define **MLME_STA_DISASSOC** 1
- #define **HOSTAP_CRYPT_FLAG_SET_TX_KEY** BIT(0)
- #define **HOSTAP_CRYPT_FLAG_PERMANENT** BIT(1)
- #define **HOSTAP_CRYPT_ERR_UNKNOWN_ALG** 2
- #define **HOSTAP_CRYPT_ERR_UNKNOWN_ADDR** 3
- #define **HOSTAP_CRYPT_ERR_CRYPT_INIT_FAILED** 4
- #define **HOSTAP_CRYPT_ERR_KEY_SET_FAILED** 5
- #define **HOSTAP_CRYPT_ERR_TX_KEY_SET_FAILED** 6
- #define **HOSTAP_CRYPT_ERR_CARD_CONF_FAILED** 7

Enumerations

- enum {

  PRISM2_PARAM_TXRATECTRL = 2, **PRISM2_PARAM_BEACON_INT** = 3, **PRISM2_-PARAM_PSEUDO_IBSS** = 4, **PRISM2_PARAM_ALC** = 5,

  PRISM2_PARAM_DUMP = 7, **PRISM2_PARAM_OTHER_AP_POLICY** = 8, **PRISM2_-PARAM_AP_MAX_INACTIVITY** = 9, **PRISM2_PARAM_AP_BRIDGE_PACKETS** = 10,

  PRISM2_PARAM_DTIM_PERIOD = 11, **PRISM2_PARAM_AP_NULLFUNC_ACK** = 12, **PRISM2_PARAM_MAX_WDS** = 13, **PRISM2_PARAM_AP_AUTOM_AP_WDS** = 14,

  PRISM2_PARAM_AP_AUTH_ALGS = 15, **PRISM2_PARAM_MONITOR_ALLOW_-FCSERR** = 16, **PRISM2_PARAM_HOST_ENCRYPT** = 17, **PRISM2_PARAM_HOST_-DECRYPT** = 18,

  PRISM2_PARAM_BUS_MASTER_THRESHOLD_RX = 19, **PRISM2_PARAM_BUS_-MASTER_THRESHOLD_TX** = 20, **PRISM2_PARAM_HOST_ROAMING** = 21, **PRISM2_-PARAM_BCRX_STA_KEY** = 22,

  PRISM2_PARAM_IEEE_802_1X = 23, **PRISM2_PARAM_ANTSEL_TX** = 24, **PRISM2_-PARAM_ANTSEL_RX** = 25, **PRISM2_PARAM_MONITOR_TYPE** = 26,

  PRISM2_PARAM_WDS_TYPE = 27, **PRISM2_PARAM_HOSTSCAN** = 28, **PRISM2_-PARAM_AP_SCAN** = 29, **PRISM2_PARAM_ENH_SEC** = 30,

  PRISM2_PARAM_IO_DEBUG = 31, **PRISM2_PARAM_BASIC_RATES** = 32, **PRISM2_-PARAM_OPER_RATES** = 33, **PRISM2_PARAM_HOSTAPD** = 34,

  PRISM2_PARAM_HOSTAPD_STA = 35, **PRISM2_PARAM_WPA** = 36, **PRISM2_PARAM_-PRIVACY_INVOKED** = 37, **PRISM2_PARAM_TKIP_COUNTERMEASURES** = 38,

  PRISM2_PARAM_DROP_UNENCRYPTED = 39, **PRISM2_PARAM_SCAN_CHANNEL_-MASK** = 40 }
- enum { **HOSTAP_ANTSEL_DO_NOT_TOUCH** = 0, **HOSTAP_ANTSEL_DIVERSITY** = 1, **HOSTAP_ANTSEL_LOW** = 2, **HOSTAP_ANTSEL_HIGH** = 3 }
- enum {

  AP_MAC_CMD_POLICY_OPEN = 0, **AP_MAC_CMD_POLICY_ALLOW** = 1, **AP_MAC_-CMD_POLICY_DENY** = 2, **AP_MAC_CMD_FLUSH** = 3,

  AP_MAC_CMD_KICKALL = 4 }
- enum {

  PRISM2_DOWNLOAD_VOLATILE = 1, **PRISM2_DOWNLOAD_NON_VOLATILE** = 3, **PRISM2_DOWNLOAD_VOLATILE_GENESIS** = 4, **PRISM2_DOWNLOAD_VOLATILE_-PERSISTENT** = 5,

  PRISM2_DOWNLOAD_VOLATILE_GENESIS_PERSISTENT = 6 }
- enum {

  PRISM2_HOSTAPD_FLUSH = 1, **PRISM2_HOSTAPD_ADD_STA** = 2, **PRISM2_-HOSTAPD_REMOVE_STA** = 3, **PRISM2_HOSTAPD_GET_INFO_STA** = 4,

  PRISM2_SET_ENCRYPTION = 6, **PRISM2_GET_ENCRYPTION** = 7, **PRISM2_-HOSTAPD_SET_FLAGS_STA** = 8, **PRISM2_HOSTAPD_GET_RID** = 9,

  PRISM2_HOSTAPD_SET_RID = 10, **PRISM2_HOSTAPD_SET_ASSOC_AP_ADDR** = 11, **PRISM2_HOSTAPD_SET_GENERIC_ELEMENT** = 12, **PRISM2_HOSTAPD_MLME** = 13,

  PRISM2_HOSTAPD_SCAN_REQ = 14, **PRISM2_HOSTAPD_STA_CLEAR_STATS** = 15 }

6.74.1 Detailed Description

hostapd / Kernel driver communication with Linux Host AP driver

Copyright

Copyright (c) 2002-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

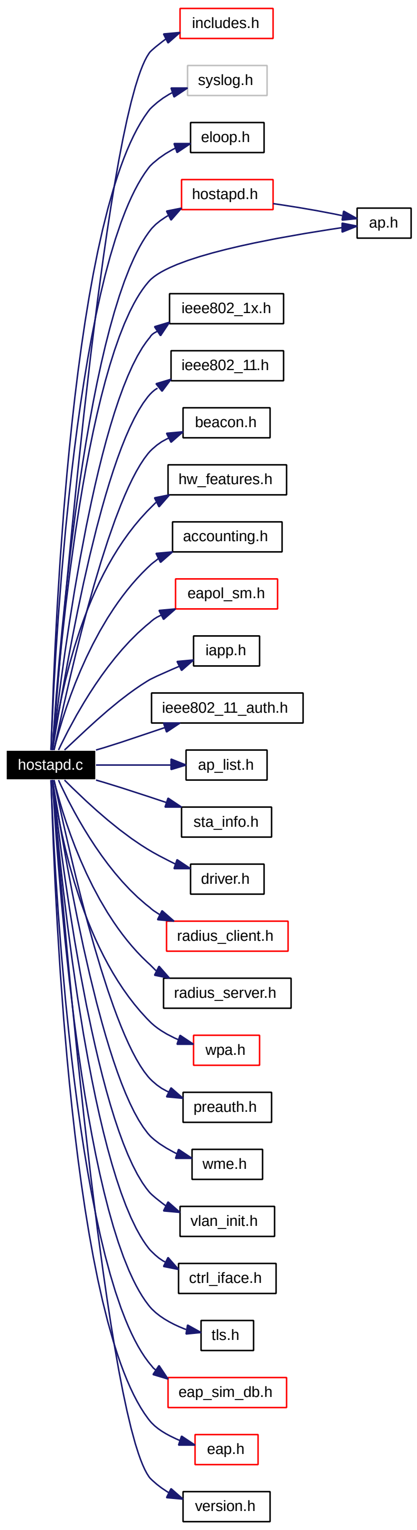
Definition in file hostap_common.h.

6.75 hostapd.c File Reference

hostapd / Initialization and configuration

```
#include "includes.h"
#include <syslog.h>
#include "eloop.h"
#include "hostapd.h"
#include "ieee802_1x.h"
#include "ieee802_11.h"
#include "beacon.h"
#include "hw_features.h"
#include "accounting.h"
#include "eapol_sm.h"
#include "iapp.h"
#include "ap.h"
#include "ieee802_11_auth.h"
#include "ap_list.h"
#include "sta_info.h"
#include "driver.h"
#include "radius_client.h"
#include "radius_server.h"
#include "wpa.h"
#include "preauth.h"
#include "wme.h"
#include "vlan_init.h"
#include "ctrl_iface.h"
#include "tls.h"
#include "eap_sim_db.h"
#include "eap.h"
#include "version.h"
```

Include dependency graph for hostapd.c:


includes.h
syslog.h
eloop.h
hostapd.h
ap.h
ieee802_1x.h
ieee802_11.h
beacon.h
hw_features.h
accounting.h
eapol_sm.h
iapp.h
ieee802_11_auth.h
hostapd.c
ap_list.h
sta_info.h
driver.h
radius_client.h
radius_server.h
wpa.h
preauth.h
wme.h
vlan_init.h
ctrl_iface.h
tls.h
eap_sim_db.h
eap.h
version.h

Functions

- void **hostapd_logger** (struct hostapd_data ∗hapd, const u8 ∗addr, unsigned int module, int level, const char ∗fmt,...)
- const char ∗ **hostapd_ip_txt** (const struct hostapd_ip_addr ∗addr, char ∗buf, size_t buflen)
- int **hostapd_ip_diff** (struct hostapd_ip_addr ∗a, struct hostapd_ip_addr ∗b)
- void hostapd_new_assoc_sta (struct hostapd_data ∗hapd, struct sta_info ∗sta, int reassoc)

  Notify that a new station associated with the AP.

- int hostapd_setup_interface_start (struct hostapd_iface ∗iface, hostapd_iface_cb cb)

  Start the setup of an interface.

- int hostapd_setup_interface_stop (struct hostapd_iface ∗iface)

  Stops the setup of an interface.

- void **driver_register** (const char ∗name, const struct driver_ops ∗ops)
- void **driver_unregister** (const char ∗name)
- const struct driver_ops ∗ **driver_lookup** (const char ∗name)
- void register_drivers (void)

  Register driver interfaces.

- int **main** (int argc, char ∗argv[ ])

Variables

- unsigned char **rfc1042_header** [6] = { 0xaa, 0xaa, 0x03, 0x00, 0x00, 0x00 }
- int **wpa_debug_level**
- int **wpa_debug_show_keys**
- int **wpa_debug_timestamp**

6.75.1 Detailed Description

hostapd / Initialization and configuration

Copyright

Copyright (c) 2002-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file hostapd.c.

6.75.2 Function Documentation

6.75.2.1 void hostapd_new_assoc_sta (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*, int *reassoc*)

Notify that a new station associated with the AP.

Parameters:

hapd Pointer to BSS data

sta Pointer to the associated STA data

reassoc 1 to indicate this was a re-association; 0 = first association

This function will be called whenever a station associates with the AP. It can be called for ieee802_11.c for drivers that export MLME to hostapd and from driver_∗.c for drivers that take care of management frames (IEEE 802.11 authentication and association) internally.

Definition at line 275 of file hostapd.c.

6.75.2.2 int hostapd_setup_interface_start (struct hostapd_iface ∗ *iface*, hostapd_iface_cb *cb*)

Start the setup of an interface.

Parameters:

iface Pointer to interface data.

cb The function to callback when done.

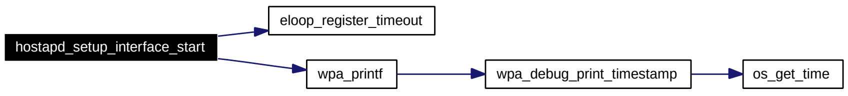
Returns:

0 if it starts successfully; cb will be called when done. -1 on failure; cb will not be called.

Initializes the driver interface, validates the configuration, and sets driver parameters based on the configuration. Flushes old stations, sets the channel, DFS parameters, encryption, beacons, and WDS links based on the configuration.

Definition at line 1475 of file hostapd.c.

Here is the call graph for this function:



6.75.2.3 int hostapd_setup_interface_stop (struct hostapd_iface ∗ *iface*)

Stops the setup of an interface.

Parameters:

iface Pointer to interface data

Returns:

0 if successfully stopped; -1 on failure (i.e., was not in progress)

Definition at line 1500 of file hostapd.c.

6.75.2.4 void register_drivers (void)

Register driver interfaces.

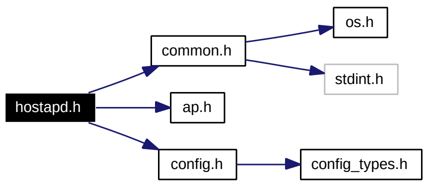
This function is generated by Makefile (into driver_conf.c) to call all configured driver interfaces to register them to core hostapd.

6.76 hostapd.h File Reference

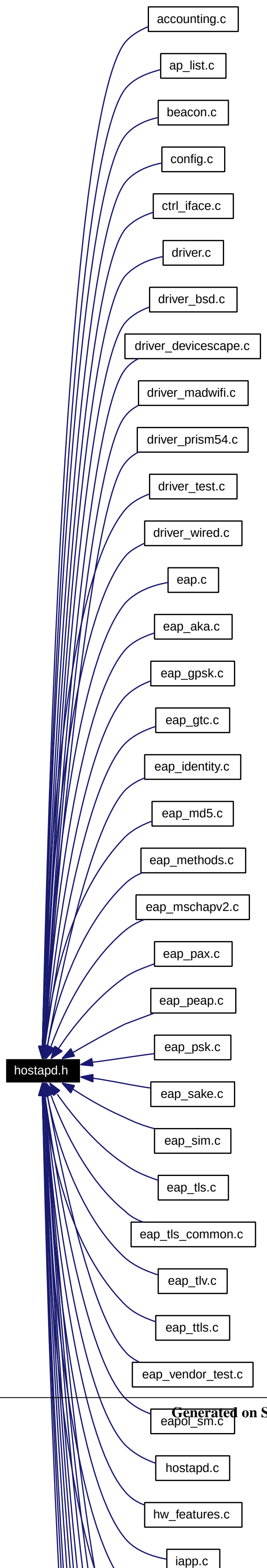
hostapd / Initialization and configuration Host AP kernel driver

```
#include "common.h"
#include "ap.h"
#include "config.h"
```

Include dependency graph for hostapd.h:



This graph shows which files directly or indirectly include this file:


accounting.c
ap_list.c
beacon.c
config.c
ctrl_iface.c
driver.c
driver_bsd.c
driver_devicescape.c
driver_madwifi.c
driver_prism54.c
driver_test.c
driver_wired.c
eap.c
eap_aka.c
eap_gpsk.c
eap_gtc.c
eap_identity.c
eap_md5.c
eap_methods.c
eap_mschapv2.c
eap_pax.c
eap_peap.c
eap_psk.c
hostapd.h
eap_sake.c
eap_sim.c
eap_tls.c
eap_tls_common.c
eap_tlv.c
eap_ttls.c
eap_vendor_test.c
eapol_sm.c
hostapd.c
hw_features.c
iapp.c

Defines

- #define **IFNAMSIZ** 16
- #define **ETH_P_ALL** 0x0003
- #define **ETH_P_PAE** 0x888E
- #define **BIT**(x) (1 << (x))
- #define **MAC2STR**(a) (a)[0], (a)[1], (a)[2], (a)[3], (a)[4], (a)[5]
- #define **MACSTR** "%02x:%02x:%02x:%02x:%02x:%02x"
- #define **MAX_VLAN_ID** 4094
- #define **IEEE80211_DA_FROMDS** addr1
- #define **IEEE80211_BSSID_FROMDS** addr2
- #define **IEEE80211_SA_FROMDS** addr3
- #define **IEEE80211_HDRLEN** (sizeof(struct ieee80211_hdr))
- #define **IEEE80211_FC**(type, stype) host_to_le16((type << 2) | (stype << 4))
- #define **HOSTAPD_MTU** 2290
- #define **HOSTAPD_DEBUG**(level, args...)
- #define **HOSTAPD_DEBUG_COND**(level) (hapd → conf → debug >= (level))

Typedefs

- typedef void(∗ hostapd_iface_cb)(struct hostapd_iface ∗iface, int status)

  Generic callback type for per-iface asynchronous requests.

Functions

- void hostapd_new_assoc_sta (struct hostapd_data ∗hapd, struct sta_info ∗sta, int reassoc)

  Notify that a new station associated with the AP.

- void **hostapd_logger** (struct hostapd_data ∗hapd, const u8 ∗addr, unsigned int module, int level, const char ∗fmt,...) PRINTF_FORMAT(5
- const char ∗ **hostapd_ip_txt** (const struct hostapd_ip_addr ∗addr, char ∗buf, size_t buflen)
- int **hostapd_ip_diff** (struct hostapd_ip_addr ∗a, struct hostapd_ip_addr ∗b)

Variables

- ieee8023_hdr **STRUCT_PACKED**
- unsigned char **rfc1042_header** [6]

6.76.1 Detailed Description

hostapd / Initialization and configuration Host AP kernel driver

Copyright

Copyright (c) 2002-2006, Jouni Malinen <jkmaline@cc.hut.fi>

Definition in file hostapd.h.

6.76.2 Define Documentation

6.76.2.1 #define HOSTAPD_DEBUG(level, args...)

Value:

```
do { \
        if (hapd->conf == NULL || hapd->conf->debug >= (level)) \
                printf(args); \
} while (0)
```

Definition at line 246 of file hostapd.h.

6.76.3 Typedef Documentation

6.76.3.1 typedef void(∗ hostapd_iface_cb)(struct hostapd_iface ∗iface, int status)

Generic callback type for per-iface asynchronous requests.

Parameters:

iface the interface the event occured on.

status 0 if the request succeeded; -1 if the request failed.

Definition at line 180 of file hostapd.h.

6.76.4 Function Documentation

6.76.4.1 void hostapd_new_assoc_sta (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*, int *reassoc*)

Notify that a new station associated with the AP.

Parameters:

hapd Pointer to BSS data

sta Pointer to the associated STA data

reassoc 1 to indicate this was a re-association; 0 = first association

This function will be called whenever a station associates with the AP. It can be called for ieee802_11.c for drivers that export MLME to hostapd and from driver_∗.c for drivers that take care of management frames (IEEE 802.11 authentication and association) internally.

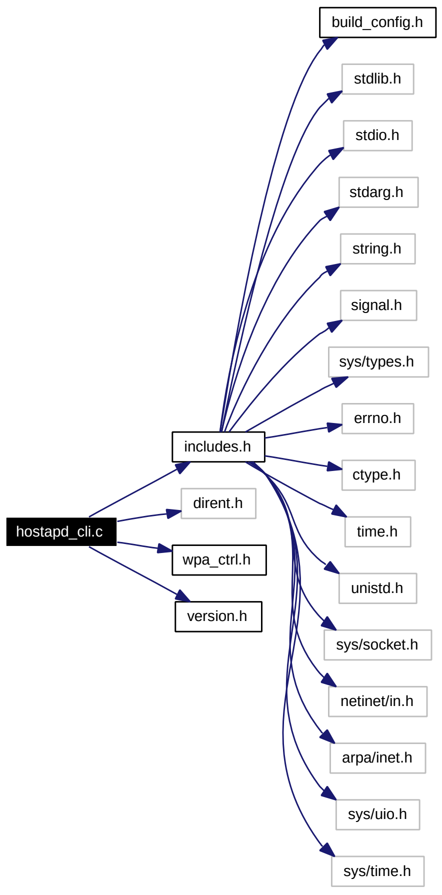
Definition at line 275 of file hostapd.c.

6.77 hostapd_cli.c File Reference

hostapd - command line interface for hostapd daemon

```
#include "includes.h"
#include <dirent.h>
#include "wpa_ctrl.h"
#include "version.h"
```

Include dependency graph for hostapd_cli.c:



Functions

- int **main** (int argc, char ∗argv[ ])

6.77.1 Detailed Description

hostapd - command line interface for hostapd daemon

Copyright

Copyright (c) 2004-2005, Jouni Malinen <jkmaline@cc.hut.fi>

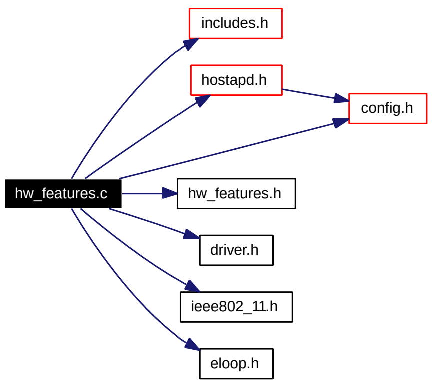
Definition in file hostapd_cli.c.

6.78 hw_features.c File Reference

hostapd / Hardware feature query and different modes

```
#include "includes.h"
#include "hostapd.h"
#include "hw_features.h"
#include "driver.h"
#include "config.h"
#include "ieee802_11.h"
#include "eloop.h"
```

Include dependency graph for hw_features.c:



Functions

- void **hostapd_free_hw_features** (struct hostapd_hw_modes *hw_features, size_t num_hw_-features)
- int **hostapd_get_hw_features** (struct hostapd_iface *iface)
- int hostapd_select_hw_mode_start (struct hostapd_iface *iface, hostapd_iface_cb cb)

  Start selection of the hardware mode.

- int hostapd_select_hw_mode_stop (struct hostapd_iface *iface)

  Stops automatic channel selection.

- const char * **hostapd_hw_mode_txt** (int mode)
- int **hostapd_hw_get_freq** (struct hostapd_data *hapd, int chan)
- int **hostapd_hw_get_channel** (struct hostapd_data *hapd, int freq)

6.78.1 Detailed Description

hostapd / Hardware feature query and different modes

Copyright

Copyright 2002-2003, Instant802 Networks, Inc. Copyright 2005-2006, Devicescape Software, Inc.

Definition in file hw_features.c.

6.78.2 Function Documentation

6.78.2.1 int hostapd_select_hw_mode_start (struct hostapd_iface ∗ *iface*, hostapd_iface_cb *cb*)

Start selection of the hardware mode.

Parameters:

iface Pointer to interface data.

cb The function to callback when done.

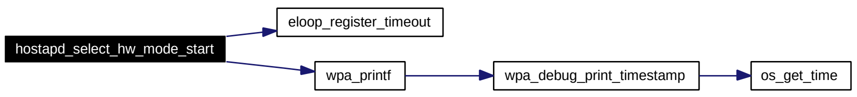
Returns:

0 if it starts successfully; cb will be called when done. -1 on failure; cb will not be called.

Sets up the hardware mode, channel, rates, and passive scanning based on the configuration.

Definition at line 358 of file hw_features.c.

Here is the call graph for this function:



6.78.2.2 int hostapd_select_hw_mode_stop (struct hostapd_iface ∗ *iface*)

Stops automatic channel selection.

Parameters:

iface Pointer to interface data.

Returns:

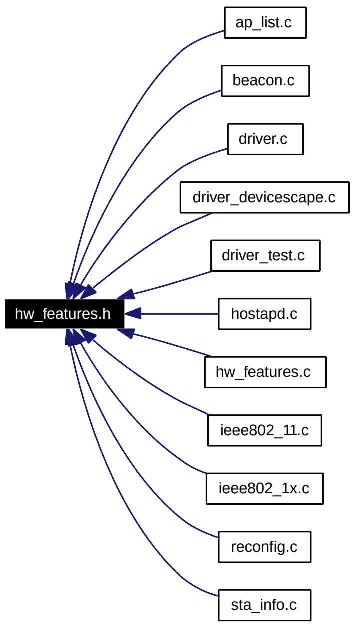
0 if successfully stopped; -1 on failure (i.e., was not in progress)

Definition at line 383 of file hw_features.c.

6.79 hw_features.h File Reference

hostapd / Hardware feature query and different modes

This graph shows which files directly or indirectly include this file:



Defines

- #define **HOSTAPD_CHAN_W_SCAN** 0x00000001
- #define **HOSTAPD_CHAN_W_ACTIVE_SCAN** 0x00000002
- #define **HOSTAPD_CHAN_W_IBSS** 0x00000004
- #define **HOSTAPD_RATE_ERP** 0x00000001
- #define **HOSTAPD_RATE_BASIC** 0x00000002
- #define **HOSTAPD_RATE_PREAMBLE2** 0x00000004
- #define **HOSTAPD_RATE_SUPPORTED** 0x00000010
- #define **HOSTAPD_RATE_OFDM** 0x00000020
- #define **HOSTAPD_RATE_CCK** 0x00000040
- #define **HOSTAPD_RATE_MANDATORY** 0x00000100

Functions

- void **hostapd_free_hw_features** (struct hostapd_hw_modes ∗hw_features, size_t num_hw_-features)
- int **hostapd_get_hw_features** (struct hostapd_iface ∗iface)
- int hostapd_select_hw_mode_start (struct hostapd_iface ∗iface, hostapd_iface_cb cb)

  Start selection of the hardware mode.

- int hostapd_select_hw_mode_stop (struct hostapd_iface ∗iface)

  Stops automatic channel selection.

- const char ∗ **hostapd_hw_mode_txt** (int mode)
- int **hostapd_hw_get_freq** (struct hostapd_data ∗hapd, int chan)
- int **hostapd_hw_get_channel** (struct hostapd_data ∗hapd, int freq)

6.79.1 Detailed Description

hostapd / Hardware feature query and different modes

Copyright

Copyright 2002-2003, Instant802 Networks, Inc. Copyright 2005-2006, Devicescape Software, Inc.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file hw_features.h.

6.79.2 Function Documentation

6.79.2.1 int hostapd_select_hw_mode_start (struct hostapd_iface ∗ *iface*, hostapd_iface_cb *cb*)

Start selection of the hardware mode.

Parameters:

iface Pointer to interface data.

cb The function to callback when done.

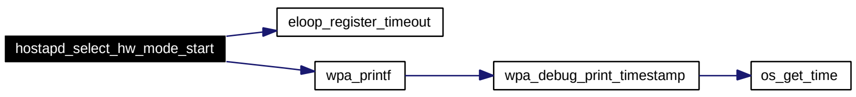
Returns:

0 if it starts successfully; cb will be called when done. -1 on failure; cb will not be called.

Sets up the hardware mode, channel, rates, and passive scanning based on the configuration.

Definition at line 358 of file hw_features.c.

Here is the call graph for this function:



6.79.2.2 int hostapd_select_hw_mode_stop (struct hostapd_iface ∗ *iface*)

Stops automatic channel selection.

Parameters:

iface Pointer to interface data.

Returns:

0 if successfully stopped; -1 on failure (i.e., was not in progress)

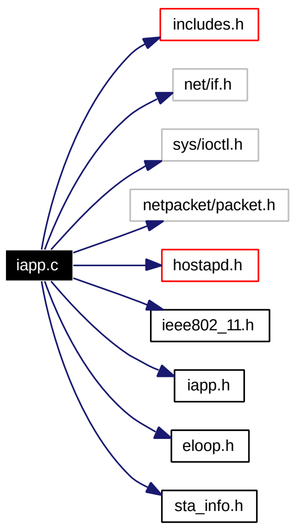
Definition at line 383 of file hw_features.c.

6.80 iapp.c File Reference

hostapd / IEEE 802.11F-2003 Inter-Access Point Protocol (IAPP)

```
#include "includes.h"
#include <net/if.h>
#include <sys/ioctl.h>
#include <netpacket/packet.h>
#include "hostapd.h"
#include "ieee802_11.h"
#include "iapp.h"
#include "eloop.h"
#include "sta_info.h"
```

Include dependency graph for iapp.c:



Defines

- #define **IAPP_MULTICAST** "224.0.1.178"
- #define **IAPP_UDP_PORT** 3517
- #define **IAPP_TCP_PORT** 3517
- #define **IAPP_VERSION** 0

Enumerations

- enum **IAPP_COMMAND** {

  IAPP_CMD_ADD_notify = 0, **IAPP_CMD_MOVE_notify** = 1, **IAPP_CMD_MOVE_response** = 2, **IAPP_CMD_Send_Security_Block** = 3,

  IAPP_CMD_ACK_Security_Block = 4, **IAPP_CMD_CACHE_notify** = 5, **IAPP_CMD_-CACHE_response** = 6 }

- enum { **IAPP_MOVE_SUCCESSFUL** = 0, **IAPP_MOVE_DENIED** = 1, **IAPP_MOVE_-STALE_MOVE** = 2 }
- enum { **IAPP_CACHE_SUCCESSFUL** = 0, **IAPP_CACHE_STALE_CACHE** = 1 }

Functions

- void **iapp_new_station** (struct iapp_data ∗iapp, struct sta_info ∗sta)
- iapp_data ∗ **iapp_init** (struct hostapd_data ∗hapd, const char ∗iface)
- void **iapp_deinit** (struct iapp_data ∗iapp)
- int **iapp_reconfig** (struct hostapd_data ∗hapd, struct hostapd_config ∗oldconf, struct hostapd_bss_-config ∗oldbss)

Variables

- iapp_hdr **packed**

6.80.1 Detailed Description

hostapd / IEEE 802.11F-2003 Inter-Access Point Protocol (IAPP)

Copyright

Copyright (c) 2002-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

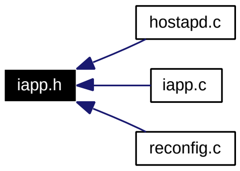
Note: IEEE 802.11F-2003 was a experimental use specification. It has expired and IEEE has withdrawn it. In other words, it is likely better to look at using some other mechanism for AP-to-AP communication than extenting the implementation here.

Definition in file iapp.c.

6.81 iapp.h File Reference

hostapd / IEEE 802.11F-2003 Inter-Access Point Protocol (IAPP)

This graph shows which files directly or indirectly include this file:



6.81.1 Detailed Description

hostapd / IEEE 802.11F-2003 Inter-Access Point Protocol (IAPP)

Copyright

Copyright (c) 2002-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

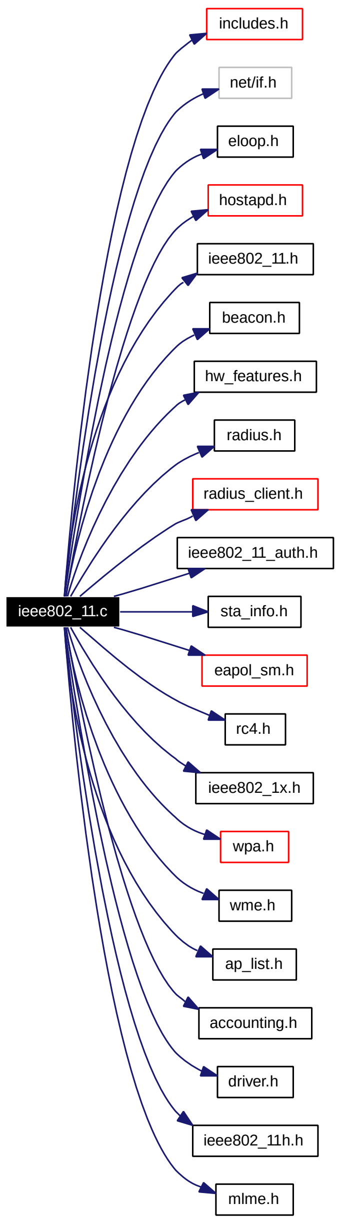
Definition in file iapp.h.

6.82 ieee802_11.c File Reference

hostapd / IEEE 802.11 Management

```
#include "includes.h"
#include <net/if.h>
#include "eloop.h"
#include "hostapd.h"
#include "ieee802_11.h"
#include "beacon.h"
#include "hw_features.h"
#include "radius.h"
#include "radius_client.h"
#include "ieee802_11_auth.h"
#include "sta_info.h"
#include "eapol_sm.h"
#include "rc4.h"
#include "ieee802_1x.h"
#include "wpa.h"
#include "wme.h"
#include "ap_list.h"
#include "accounting.h"
#include "driver.h"
#include "ieee802_11h.h"
#include "mlme.h"
```

Include dependency graph for ieee802_11.c:



Defines

- #define **OUI_MICROSOFT** 0x0050f2

Functions

- u8 ∗ **hostapd_eid_supp_rates** (struct hostapd_data ∗hapd, u8 ∗eid)
- u8 ∗ **hostapd_eid_ext_supp_rates** (struct hostapd_data ∗hapd, u8 ∗eid)
- u16 **hostapd_own_capab_info** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int probe)
- ParseRes **ieee802_11_parse_elems** (struct hostapd_data ∗hapd, u8 ∗start, size_t len, struct ieee802_-11_elems ∗elems, int show_errors)
- void **ieee802_11_print_ssid** (const u8 ∗ssid, u8 len)

- void **ieee802_11_send_deauth** (struct hostapd_data ∗hapd, u8 ∗addr, u16 reason)
- void ieee802_11_mgmt (struct hostapd_data ∗hapd, u8 ∗buf, size_t len, u16 stype, struct hostapd_-frame_info ∗fi)

  process incoming IEEE 802.11 management frames

- void **ieee802_11_mgmt_cb** (struct hostapd_data ∗hapd, u8 ∗buf, size_t len, u16 stype, int ok)
- void **ieee80211_michael_mic_failure** (struct hostapd_data ∗hapd, const u8 ∗addr, int local)
- int **ieee802_11_get_mib** (struct hostapd_data ∗hapd, char ∗buf, size_t buflen)
- int **ieee802_11_get_mib_sta** (struct hostapd_data ∗hapd, struct sta_info ∗sta, char ∗buf, size_t bu-flen)

6.82.1 Detailed Description

hostapd / IEEE 802.11 Management

Copyright

Definition in file ieee802_11.c.

6.82.2 Function Documentation

6.82.2.1 void ieee802_11_mgmt (struct hostapd_data ∗ *hapd*, u8 ∗ *buf*, size_t *len*, u16 *stype*, struct hostapd_frame_info ∗ *fi*)

process incoming IEEE 802.11 management frames

Parameters:

hapd hostapd BSS data structure (the BSS to which the management frame was sent to)

buf management frame data (starting from IEEE 802.11 header)

len length of frame data in octets

stype management frame subtype from frame control field

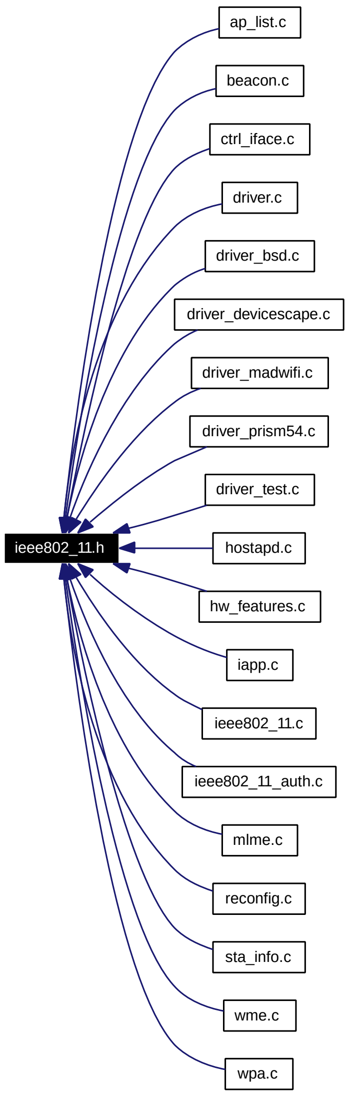
Process all incoming IEEE 802.11 management frames. This will be called for each frame received from the kernel driver through wlan#ap interface. In addition, it can be called to re-inserted pending frames (e.g., when using external RADIUS server as an MAC ACL).

Definition at line 1289 of file ieee802_11.c.

6.83 ieee802_11.h File Reference

hostapd / IEEE 802.11 Management

This graph shows which files directly or indirectly include this file:



Defines

- #define **WLAN_FC_PVER** (BIT(1) | BIT(0))
- #define **WLAN_FC_TODS** BIT(8)
- #define **WLAN_FC_FROMDS** BIT(9)
- #define **WLAN_FC_MOREFRAG** BIT(10)
- #define **WLAN_FC_RETRY** BIT(11)
- #define **WLAN_FC_PWRMGT** BIT(12)
- #define **WLAN_FC_MOREDATA** BIT(13)
- #define **WLAN_FC_ISWEP** BIT(14)

- #define **WLAN_FC_ORDER** BIT(15)
- #define **WLAN_FC_GET_TYPE**(fc) (((fc) & (BIT(3) | BIT(2))) >> 2)
- #define **WLAN_FC_GET_STYPE**(fc) (((fc) & (BIT(7) | BIT(6) | BIT(5) | BIT(4))) >> 4)
- #define **WLAN_GET_SEQ_FRAG**(seq) ((seq) & (BIT(3) | BIT(2) | BIT(1) | BIT(0)))
- #define **WLAN_GET_SEQ_SEQ**(seq) (((seq) & (∼(BIT(3) | BIT(2) | BIT(1) | BIT(0)))) >> 4)
- #define **WLAN_FC_TYPE_MGMT** 0
- #define **WLAN_FC_TYPE_CTRL** 1
- #define **WLAN_FC_TYPE_DATA** 2
- #define **WLAN_FC_STYPE_ASSOC_REQ** 0
- #define **WLAN_FC_STYPE_ASSOC_RESP** 1
- #define **WLAN_FC_STYPE_REASSOC_REQ** 2
- #define **WLAN_FC_STYPE_REASSOC_RESP** 3
- #define **WLAN_FC_STYPE_PROBE_REQ** 4
- #define **WLAN_FC_STYPE_PROBE_RESP** 5
- #define **WLAN_FC_STYPE_BEACON** 8
- #define **WLAN_FC_STYPE_ATIM** 9
- #define **WLAN_FC_STYPE_DISASSOC** 10
- #define **WLAN_FC_STYPE_AUTH** 11
- #define **WLAN_FC_STYPE_DEAUTH** 12
- #define **WLAN_FC_STYPE_ACTION** 13
- #define **WLAN_FC_STYPE_PSPOLL** 10
- #define **WLAN_FC_STYPE_RTS** 11
- #define **WLAN_FC_STYPE_CTS** 12
- #define **WLAN_FC_STYPE_ACK** 13
- #define **WLAN_FC_STYPE_CFEND** 14
- #define **WLAN_FC_STYPE_CFENDACK** 15
- #define **WLAN_FC_STYPE_DATA** 0
- #define **WLAN_FC_STYPE_DATA_CFACK** 1
- #define **WLAN_FC_STYPE_DATA_CFPOLL** 2
- #define **WLAN_FC_STYPE_DATA_CFACKPOLL** 3
- #define **WLAN_FC_STYPE_NULLFUNC** 4
- #define **WLAN_FC_STYPE_CFACK** 5
- #define **WLAN_FC_STYPE_CFPOLL** 6
- #define **WLAN_FC_STYPE_CFACKPOLL** 7
- #define **WLAN_FC_STYPE_QOS_DATA** 8
- #define **WLAN_AUTH_OPEN** 0
- #define **WLAN_AUTH_SHARED_KEY** 1
- #define **WLAN_AUTH_CHALLENGE_LEN** 128
- #define **WLAN_CAPABILITY_ESS** BIT(0)
- #define **WLAN_CAPABILITY_IBSS** BIT(1)
- #define **WLAN_CAPABILITY_CF_POLLABLE** BIT(2)
- #define **WLAN_CAPABILITY_CF_POLL_REQUEST** BIT(3)
- #define **WLAN_CAPABILITY_PRIVACY** BIT(4)
- #define **WLAN_CAPABILITY_SHORT_PREAMBLE** BIT(5)
- #define **WLAN_CAPABILITY_PBCC** BIT(6)
- #define **WLAN_CAPABILITY_CHANNEL_AGILITY** BIT(7)
- #define **WLAN_CAPABILITY_SPECTRUM_MGMT** BIT(8)
- #define **WLAN_CAPABILITY_SHORT_SLOT_TIME** BIT(10)
- #define **WLAN_CAPABILITY_DSSS_OFDM** BIT(13)
- #define **WLAN_STATUS_SUCCESS** 0

- #define **WLAN_STATUS_UNSPECIFIED_FAILURE** 1
- #define **WLAN_STATUS_CAPS_UNSUPPORTED** 10
- #define **WLAN_STATUS_REASSOC_NO_ASSOC** 11
- #define **WLAN_STATUS_ASSOC_DENIED_UNSPEC** 12
- #define **WLAN_STATUS_NOT_SUPPORTED_AUTH_ALG** 13
- #define **WLAN_STATUS_UNKNOWN_AUTH_TRANSACTION** 14
- #define **WLAN_STATUS_CHALLENGE_FAIL** 15
- #define **WLAN_STATUS_AUTH_TIMEOUT** 16
- #define **WLAN_STATUS_AP_UNABLE_TO_HANDLE_NEW_STA** 17
- #define **WLAN_STATUS_ASSOC_DENIED_RATES** 18
- #define **WLAN_STATUS_ASSOC_DENIED_NOSHORT** 19
- #define **WLAN_STATUS_ASSOC_DENIED_NOPBCC** 20
- #define **WLAN_STATUS_ASSOC_DENIED_NOAGILITY** 21
- #define **WLAN_STATUS_SPEC_MGMT_REQUIRED** 22
- #define **WLAN_STATUS_PWR_CAPABILITY_NOT_VALID** 23
- #define **WLAN_STATUS_SUPPORTED_CHANNEL_NOT_VALID** 24
- #define **WLAN_STATUS_INVALID_IE** 40
- #define **WLAN_STATUS_GROUP_CIPHER_NOT_VALID** 41
- #define **WLAN_STATUS_PAIRWISE_CIPHER_NOT_VALID** 42
- #define **WLAN_STATUS_AKMP_NOT_VALID** 43
- #define **WLAN_STATUS_UNSUPPORTED_RSN_IE_VERSION** 44
- #define **WLAN_STATUS_INVALID_RSN_IE_CAPAB** 45
- #define **WLAN_STATUS_CIPHER_REJECTED_PER_POLICY** 46
- #define **WLAN_REASON_UNSPECIFIED** 1
- #define **WLAN_REASON_PREV_AUTH_NOT_VALID** 2
- #define **WLAN_REASON_DEAUTH_LEAVING** 3
- #define **WLAN_REASON_DISASSOC_DUE_TO_INACTIVITY** 4
- #define **WLAN_REASON_DISASSOC_AP_BUSY** 5
- #define **WLAN_REASON_CLASS2_FRAME_FROM_NONAUTH_STA** 6
- #define **WLAN_REASON_CLASS3_FRAME_FROM_NONASSOC_STA** 7
- #define **WLAN_REASON_DISASSOC_STA_HAS_LEFT** 8
- #define **WLAN_REASON_STA_REQ_ASSOC_WITHOUT_AUTH** 9
- #define **WLAN_REASON_INVALID_IE** 13
- #define **WLAN_REASON_MICHAEL_MIC_FAILURE** 14
- #define **WLAN_REASON_4WAY_HANDSHAKE_TIMEOUT** 15
- #define **WLAN_REASON_GROUP_KEY_UPDATE_TIMEOUT** 16
- #define **WLAN_REASON_IE_IN_4WAY_DIFFERS** 17
- #define **WLAN_REASON_GROUP_CIPHER_NOT_VALID** 18
- #define **WLAN_REASON_PAIRWISE_CIPHER_NOT_VALID** 19
- #define **WLAN_REASON_AKMP_NOT_VALID** 20
- #define **WLAN_REASON_UNSUPPORTED_RSN_IE_VERSION** 21
- #define **WLAN_REASON_INVALID_RSN_IE_CAPAB** 22
- #define **WLAN_REASON_IEEE_802_1X_AUTH_FAILED** 23
- #define **WLAN_REASON_CIPHER_SUITE_REJECTED** 24
- #define **WLAN_EID_SSID** 0
- #define **WLAN_EID_SUPP_RATES** 1
- #define **WLAN_EID_FH_PARAMS** 2
- #define **WLAN_EID_DS_PARAMS** 3
- #define **WLAN_EID_CF_PARAMS** 4
- #define **WLAN_EID_TIM** 5

- #define **WLAN_EID_IBSS_PARAMS** 6
- #define **WLAN_EID_COUNTRY** 7
- #define **WLAN_EID_CHALLENGE** 16
- #define **WLAN_EID_PWR_CONSTRAINT** 32
- #define **WLAN_EID_PWR_CAPABILITY** 33
- #define **WLAN_EID_TPC_REQUEST** 34
- #define **WLAN_EID_TPC_REPORT** 35
- #define **WLAN_EID_SUPPORTED_CHANNELS** 36
- #define **WLAN_EID_CHANNEL_SWITCH** 37
- #define **WLAN_EID_MEASURE_REQUEST** 38
- #define **WLAN_EID_MEASURE_REPORT** 39
- #define **WLAN_EID_QUITE** 40
- #define **WLAN_EID_IBSS_DFS** 41
- #define **WLAN_EID_ERP_INFO** 42
- #define **WLAN_EID_RSN** 48
- #define **WLAN_EID_EXT_SUPP_RATES** 50
- #define **WLAN_EID_GENERIC** 221
- #define **WLAN_EID_VENDOR_SPECIFIC** 221
- #define **ERP_INFO_NON_ERP_PRESENT** BIT(0)
- #define **ERP_INFO_USE_PROTECTION** BIT(1)
- #define **ERP_INFO_BARKER_PREAMBLE_MODE** BIT(2)

Enumerations

- enum **ParseRes** { **ParseOK** = 0, **ParseUnknown** = 1, **ParseFailed** = -1 }

Functions

- void **ieee802_11_send_deauth** (struct hostapd_data ∗hapd, u8 ∗addr, u16 reason)
- void ieee802_11_mgmt (struct hostapd_data ∗hapd, u8 ∗buf, size_t len, u16 stype, struct hostapd_-frame_info ∗fi)

  process incoming IEEE 802.11 management frames

- void **ieee802_11_mgmt_cb** (struct hostapd_data ∗hapd, u8 ∗buf, size_t len, u16 stype, int ok)
- ParseRes **ieee802_11_parse_elems** (struct hostapd_data ∗hapd, u8 ∗start, size_t len, struct ieee802_-11_elems ∗elems, int show_errors)
- void **ieee802_11_print_ssid** (const u8 ∗ssid, u8 len)
- void **ieee80211_michael_mic_failure** (struct hostapd_data ∗hapd, const u8 ∗addr, int local)
- int **ieee802_11_get_mib** (struct hostapd_data ∗hapd, char ∗buf, size_t buflen)
- int **ieee802_11_get_mib_sta** (struct hostapd_data ∗hapd, struct sta_info ∗sta, char ∗buf, size_t bu-flen)
- u16 **hostapd_own_capab_info** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int probe)
- u8 ∗ **hostapd_eid_supp_rates** (struct hostapd_data ∗hapd, u8 ∗eid)
- u8 ∗ **hostapd_eid_ext_supp_rates** (struct hostapd_data ∗hapd, u8 ∗eid)

Variables

- ieee80211_mgmt **packed**

6.83.1 Detailed Description

hostapd / IEEE 802.11 Management

Definition in file ieee802_11.h.

6.83.2 Function Documentation

6.83.2.1 void ieee802_11_mgmt (struct hostapd_data * *hapd*, u8 * *buf*, size_t *len*, u16 *stype*, struct hostapd_frame_info * *fi*)

process incoming IEEE 802.11 management frames

Parameters:

hapd hostapd BSS data structure (the BSS to which the management frame was sent to)

buf management frame data (starting from IEEE 802.11 header)

len length of frame data in octets

stype management frame subtype from frame control field

Process all incoming IEEE 802.11 management frames. This will be called for each frame received from the kernel driver through wlan#ap interface. In addition, it can be called to re-inserted pending frames (e.g., when using external RADIUS server as an MAC ACL).

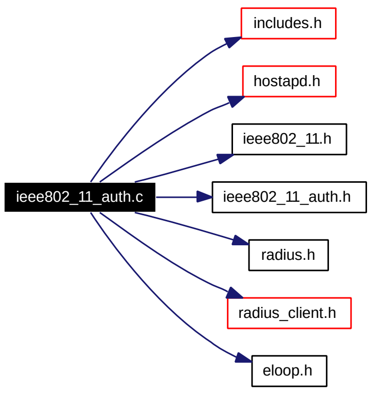
Definition at line 1289 of file ieee802_11.c.

6.84 ieee802_11_auth.c File Reference

hostapd / IEEE 802.11 authentication (ACL)

```
#include "includes.h"
#include "hostapd.h"
#include "ieee802_11.h"
#include "ieee802_11_auth.h"
#include "radius.h"
#include "radius_client.h"
#include "eloop.h"
```

Include dependency graph for ieee802_11_auth.c:



Defines

- #define **RADIUS_ACL_TIMEOUT** 30

Functions

- int **hostapd_allowed_address** (struct hostapd_data ∗hapd, const u8 ∗addr, const u8 ∗msg, size_t len, u32 ∗session_timeout, u32 ∗acct_interim_interval, int ∗vlan_id)
- int **hostapd_acl_init** (struct hostapd_data ∗hapd)
- void **hostapd_acl_deinit** (struct hostapd_data ∗hapd)
- int **hostapd_acl_reconfig** (struct hostapd_data ∗hapd, struct hostapd_config ∗oldconf)

6.84.1 Detailed Description

hostapd / IEEE 802.11 authentication (ACL)

Copyright

Copyright (c) 2003-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

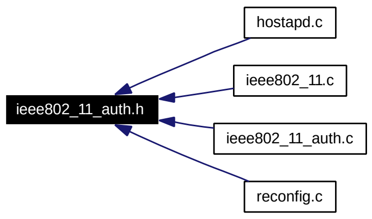
See README and COPYING for more details.

Definition in file ieee802_11_auth.c.

6.85 ieee802_11_auth.h File Reference

hostapd / IEEE 802.11 authentication (ACL)

This graph shows which files directly or indirectly include this file:



Enumerations

- enum { **HOSTAPD_ACL_REJECT** = 0, **HOSTAPD_ACL_ACCEPT** = 1, **HOSTAPD_ACL_-PENDING** = 2, **HOSTAPD_ACL_ACCEPT_TIMEOUT** = 3 }

Functions

- int **hostapd_allowed_address** (struct hostapd_data ∗hapd, const u8 ∗addr, const u8 ∗msg, size_t len, u32 ∗session_timeout, u32 ∗acct_interim_interval, int ∗vlan_id)
- int **hostapd_acl_init** (struct hostapd_data ∗hapd)
- void **hostapd_acl_deinit** (struct hostapd_data ∗hapd)
- int **hostapd_acl_reconfig** (struct hostapd_data ∗hapd, struct hostapd_config ∗oldconf)

6.85.1 Detailed Description

hostapd / IEEE 802.11 authentication (ACL)

Copyright

Definition in file ieee802_11_auth.h.

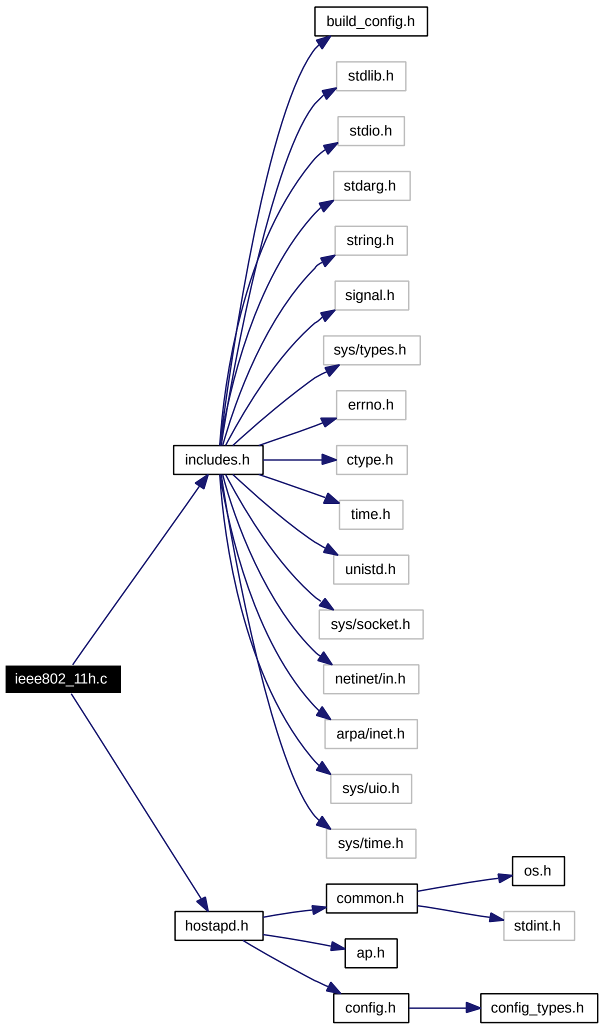
6.86 ieee802_11h.c File Reference

hostapd / IEEE 802.11h

`#include "includes.h"`

`#include "hostapd.h"`

Include dependency graph for ieee802_11h.c:



Functions

- int **hostapd_check_power_cap** (struct hostapd_data ∗hapd, u8 ∗power, u8 len)

6.86.1 Detailed Description

hostapd / IEEE 802.11h

Copyright

Copyright 2005-2006, Devicescape Software, Inc.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

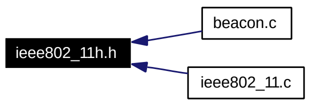
See README and COPYING for more details.

Definition in file ieee802_11h.c.

6.87 ieee802_11h.h File Reference

hostapd / IEEE 802.11h

This graph shows which files directly or indirectly include this file:



Defines

- #define **SPECT_LOOSE_BINDING** 1
- #define **SPECT_STRICT_BINDING** 2
- #define **CHAN_SWITCH_MODE_NOISY** 0
- #define **CHAN_SWITCH_MODE_QUIET** 1

Functions

- int **hostapd_check_power_cap** (struct hostapd_data ∗hapd, u8 ∗power, u8 len)

6.87.1 Detailed Description

hostapd / IEEE 802.11h

Copyright

Copyright 2005-2006, Devicescape Software, Inc. All Rights Reserved.

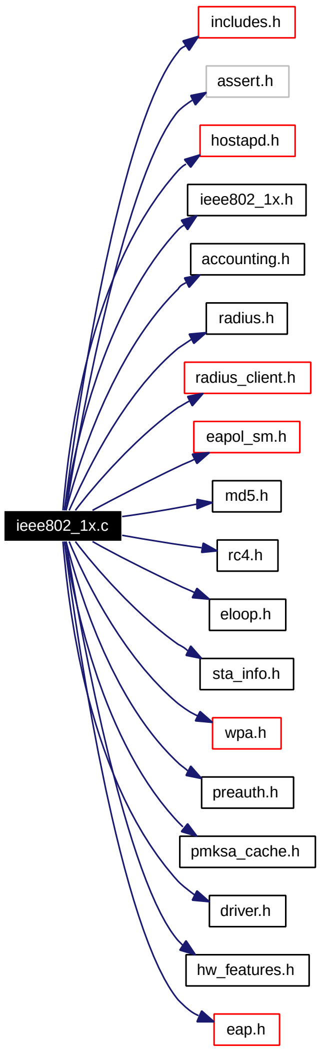
Definition in file ieee802_11h.h.

6.88 ieee802_1x.c File Reference

hostapd / IEEE 802.1X Authenticator

```
#include "includes.h"
#include <assert.h>
#include "hostapd.h"
#include "ieee802_1x.h"
#include "accounting.h"
#include "radius.h"
#include "radius_client.h"
#include "eapol_sm.h"
#include "md5.h"
#include "rc4.h"
#include "eloop.h"
#include "sta_info.h"
#include "wpa.h"
#include "preauth.h"
#include "pmksa_cache.h"
#include "driver.h"
#include "hw_features.h"
#include "eap.h"
```

Include dependency graph for ieee802_1x.c:



Functions

- void **ieee802_1x_set_sta_authorized** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int authorized)
- void **ieee802_1x_request_identity** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ieee802_1x_tx_canned_eap** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int success)
- void **ieee802_1x_tx_req** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ieee802_1x_tx_key** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- const char ∗ **radius_mode_txt** (struct hostapd_data ∗hapd)
- int **radius_sta_rate** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- char ∗ **eap_type_text** (u8 type)
- void **ieee802_1x_receive** (struct hostapd_data ∗hapd, const u8 ∗sa, const u8 ∗buf, size_t len)
- void **ieee802_1x_new_station** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ieee802_1x_free_radius_class** (struct radius_class_data ∗class)
- int **ieee802_1x_copy_radius_class** (struct radius_class_data ∗dst, struct radius_class_data ∗src)
- void **ieee802_1x_free_station** (struct sta_info ∗sta)
- void **ieee802_1x_send_resp_to_server** (struct hostapd_data ∗hapd, struct sta_info ∗sta)

- void **ieee802_1x_abort_auth** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- int **ieee802_1x_init** (struct hostapd_data ∗hapd)
- void **ieee802_1x_deinit** (struct hostapd_data ∗hapd)
- int **ieee802_1x_reconfig** (struct hostapd_data ∗hapd, struct hostapd_config ∗oldconf, struct hostapd_bss_config ∗oldbss)
- int **ieee802_1x_tx_status** (struct hostapd_data ∗hapd, struct sta_info ∗sta, u8 ∗buf, size_t len, int ack)
- u8 ∗ **ieee802_1x_get_identity** (struct eapol_state_machine ∗sm, size_t ∗len)
- u8 ∗ **ieee802_1x_get_radius_class** (struct eapol_state_machine ∗sm, size_t ∗len, int idx)
- u8 ∗ **ieee802_1x_get_key_crypt** (struct eapol_state_machine ∗sm, size_t ∗len)
- void **ieee802_1x_notify_port_enabled** (struct eapol_state_machine ∗sm, int enabled)
- void **ieee802_1x_notify_port_valid** (struct eapol_state_machine ∗sm, int valid)
- void **ieee802_1x_notify_pre_auth** (struct eapol_state_machine ∗sm, int pre_auth)
- int **ieee802_1x_get_mib** (struct hostapd_data ∗hapd, char ∗buf, size_t buflen)
- int **ieee802_1x_get_mib_sta** (struct hostapd_data ∗hapd, struct sta_info ∗sta, char ∗buf, size_t buflen)
- void **ieee802_1x_finished** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int success)

6.88.1 Detailed Description

hostapd / IEEE 802.1X Authenticator

Copyright

Copyright (c) 2002-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

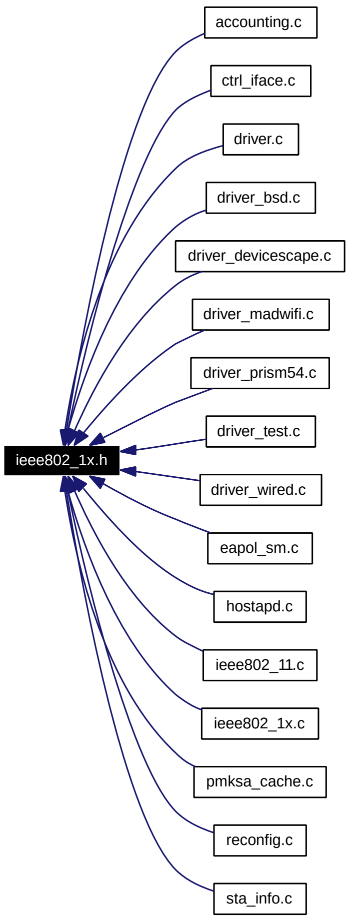
See README and COPYING for more details.

Definition in file ieee802_1x.c.

6.89 ieee802_1x.h File Reference

hostapd / IEEE 802.1X Authenticator

This graph shows which files directly or indirectly include this file:



Functions

- void **ieee802_1x_receive** (struct hostapd_data ∗hapd, const u8 ∗sa, const u8 ∗buf, size_t len)
- void **ieee802_1x_new_station** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ieee802_1x_free_station** (struct sta_info ∗sta)
- void **ieee802_1x_request_identity** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ieee802_1x_tx_canned_eap** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int success)
- void **ieee802_1x_tx_req** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ieee802_1x_tx_key** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ieee802_1x_send_resp_to_server** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ieee802_1x_abort_auth** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ieee802_1x_set_sta_authorized** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int authorized)
- void **ieee802_1x_dump_state** (FILE ∗f, const char ∗prefix, struct sta_info ∗sta)
- int **ieee802_1x_init** (struct hostapd_data ∗hapd)

- void **ieee802_1x_deinit** (struct hostapd_data ∗hapd)
- int **ieee802_1x_reconfig** (struct hostapd_data ∗hapd, struct hostapd_config ∗oldconf, struct hostapd_bss_config ∗oldbss)
- int **ieee802_1x_tx_status** (struct hostapd_data ∗hapd, struct sta_info ∗sta, u8 ∗buf, size_t len, int ack)
- u8 ∗ **ieee802_1x_get_identity** (struct eapol_state_machine ∗sm, size_t ∗len)
- u8 ∗ **ieee802_1x_get_radius_class** (struct eapol_state_machine ∗sm, size_t ∗len, int idx)
- u8 ∗ **ieee802_1x_get_key_crypt** (struct eapol_state_machine ∗sm, size_t ∗len)
- void **ieee802_1x_notify_port_enabled** (struct eapol_state_machine ∗sm, int enabled)
- void **ieee802_1x_notify_port_valid** (struct eapol_state_machine ∗sm, int valid)
- void **ieee802_1x_notify_pre_auth** (struct eapol_state_machine ∗sm, int pre_auth)
- int **ieee802_1x_get_mib** (struct hostapd_data ∗hapd, char ∗buf, size_t buflen)
- int **ieee802_1x_get_mib_sta** (struct hostapd_data ∗hapd, struct sta_info ∗sta, char ∗buf, size_t buflen)
- void **hostapd_get_ntp_timestamp** (u8 ∗buf)
- void **ieee802_1x_finished** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int success)
- char ∗ **eap_type_text** (u8 type)
- void **ieee802_1x_free_radius_class** (struct radius_class_data ∗class)
- int **ieee802_1x_copy_radius_class** (struct radius_class_data ∗dst, struct radius_class_data ∗src)

Variables

- ieee802_1x_eapol_key **packed**

6.89.1 Detailed Description

hostapd / IEEE 802.1X Authenticator

Copyright

Copyright (c) 2002-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

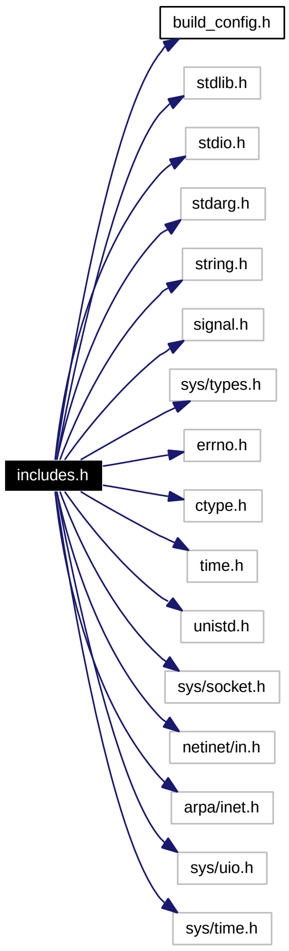
Definition in file ieee802_1x.h.

6.90 includes.h File Reference

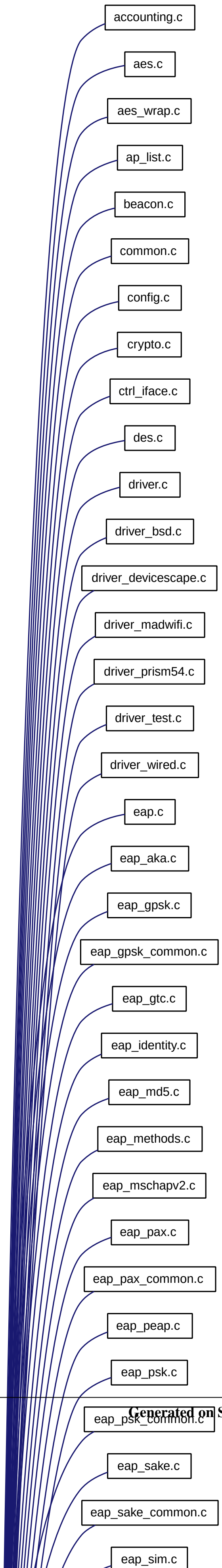
wpa_supplicant/hostapd - Default include files

```
#include "build_config.h"
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <signal.h>
#include <sys/types.h>
#include <errno.h>
#include <ctype.h>
#include <time.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/uio.h>
#include <sys/time.h>
```

Include dependency graph for includes.h:



This graph shows which files directly or indirectly include this file:


accounting.c
aes.c
aes_wrap.c
ap_list.c
beacon.c
common.c
config.c
crypto.c
ctrl_iface.c
des.c
driver.c
driver_bsd.c
driver_devicescape.c
driver_madwifi.c
driver_prism54.c
driver_test.c
driver_wired.c
eap.c
eap_aka.c
eap_gpsk.c
eap_gpsk_common.c
eap_gtc.c
eap_identity.c
eap_md5.c
eap_methods.c
eap_mschapv2.c
eap_pax.c
eap_pax_common.c
eap_peap.c
eap_psk.c
eap_psk_common.c
eap_sake.c
eap_sake_common.c
eap_sim.c

6.90.1 Detailed Description

wpa_supplicant/hostapd - Default include files

Copyright

Copyright (c) 2005-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

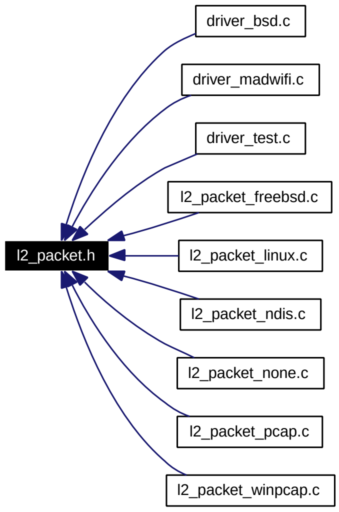
This header file is included into all C files so that commonly used header files can be selected with OS specific #ifdefs in one place instead of having to have OS/C library specific selection in many files.

Definition in file includes.h.

6.91 l2_packet.h File Reference

WPA Supplicant - Layer2 packet interface definition.

This graph shows which files directly or indirectly include this file:



Defines

- #define **MAC2STR**(a) (a)[0], (a)[1], (a)[2], (a)[3], (a)[4], (a)[5]
- #define **MACSTR** "%02x:%02x:%02x:%02x:%02x:%02x"
- #define **ETH_P_EAPOL** 0x888e
- #define **ETH_P_RSN_PREAUTH** 0x88c7

Functions

- l2_packet_data ∗ l2_packet_init (const char ∗ifname, const u8 ∗own_addr, unsigned short protocol, void(∗rx_callback)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len), void ∗rx_callback_ctx, int l2_hdr)

  Initialize l2_packet interface.

- void l2_packet_deinit (struct l2_packet_data ∗l2)

  Deinitialize l2_packet interface.

- int l2_packet_get_own_addr (struct l2_packet_data ∗l2, u8 ∗addr)

  Get own layer 2 address.

- int l2_packet_send (struct l2_packet_data ∗l2, const u8 ∗dst_addr, u16 proto, const u8 ∗buf, size_t len)

  Send a packet.

- int l2_packet_get_ip_addr (struct l2_packet_data ∗l2, char ∗buf, size_t len)

  Get the current IP address from the interface.

- void l2_packet_notify_auth_start (struct l2_packet_data *l2)

    Notify l2_packet about start of authentication.

Variables

- l2_ethhdr **STRUCT_PACKED**

6.91.1 Detailed Description

WPA Supplicant - Layer2 packet interface definition.

Copyright

Copyright (c) 2003-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

This file defines an interface for layer 2 (link layer) packet sending and receiving. l2_packet_linux.c is one implementation for such a layer 2 implementation using Linux packet sockets and l2_packet_pcap.c another one using libpcap and libdnet. When porting wpa_supplicant to other operating systems, a new l2_packet implementation may need to be added.

Definition in file l2_packet.h.

6.91.2 Function Documentation

6.91.2.1 void l2_packet_deinit (struct l2_packet_data * *l2*)

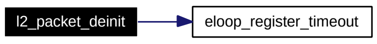
Deinitialize l2_packet interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

Definition at line 225 of file l2_packet_freebsd.c.

Here is the call graph for this function:



6.91.2.2 int l2_packet_get_ip_addr (struct l2_packet_data * *l2*, char * *buf*, size_t *len*)

Get the current IP address from the interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

buf Buffer for the IP address in text format

len Maximum buffer length

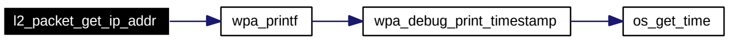
Returns:

0 on success, -1 on failure

This function can be used to get the current IP address from the interface bound to the l2_packet. This is mainly for status information and the IP address will be stored as an ASCII string. This function is not essential for wpa_supplicant operation, so full implementation is not required. l2_packet implementation will need to define the function, but it can return -1 if the IP address information is not available.

Definition at line 235 of file l2_packet_freebsd.c.

Here is the call graph for this function:



6.91.2.3 int l2_packet_get_own_addr (struct l2_packet_data ∗ *l2*, u8 ∗ *addr*)

Get own layer 2 address.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

addr Buffer for the own address (6 bytes)

Returns:

0 on success, -1 on failure

Definition at line 45 of file l2_packet_freebsd.c.

6.91.2.4 struct l2_packet_data∗ l2_packet_init (const char ∗ *ifname*, const u8 ∗ *own_addr*, unsigned short *protocol*, void(∗)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len) *rx_callback*, void ∗ *rx_callback_ctx*, int *l2_hdr*)

Initialize l2_packet interface.

Parameters:

ifname Interface name

own_addr Optional own MAC address if available from driver interface or NULL if not available

protocol Ethernet protocol number in host byte order

rx_callback Callback function that will be called for each received packet

rx_callback_ctx Callback data (ctx) for calls to rx_callback()

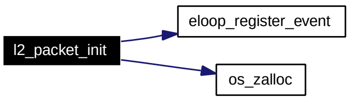
l2_hdr 1 = include layer 2 header, 0 = do not include header

Returns:

Pointer to internal data or NULL on failure rx_callback function will be called with src_addr pointing to the source address (MAC address) of the the packet. If l2_hdr is set to 0, buf points to len bytes of the payload after the layer 2 header and similarly, TX buffers start with payload. This behavior can be changed by setting l2_hdr=1 to include the layer 2 header in the data buffer.

Definition at line 193 of file l2_packet_freebsd.c.

Here is the call graph for this function:



6.91.2.5 void l2_packet_notify_auth_start (struct l2_packet_data ∗ *l2*)

Notify l2_packet about start of authentication.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

This function is called when authentication is expected to start, e.g., when association has been completed, in order to prepare l2_packet implementation for EAPOL frames. This function is used mainly if the l2_-packet code needs to do polling in which case it can increasing polling frequency. This can also be an empty function if the l2_packet implementation does not benefit from knowing about the starting authentication.

Definition at line 271 of file l2_packet_freebsd.c.

6.91.2.6 int l2_packet_send (struct l2_packet_data ∗ *l2*, const u8 ∗ *dst_addr*, u16 *proto*, const u8 ∗ *buf*, size_t *len*)

Send a packet.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

dst_addr Destination address for the packet (only used if l2_hdr == 0)

proto Protocol/ethertype for the packet in host byte order (only used if l2_hdr == 0)

buf Packet contents to be sent; including layer 2 header if l2_hdr was set to 1 in l2_packet_init() call. Otherwise, only the payload of the packet is included.

len Length of the buffer (including l2 header only if l2_hdr == 1)

Returns:

>=0 on success, <0 on failure

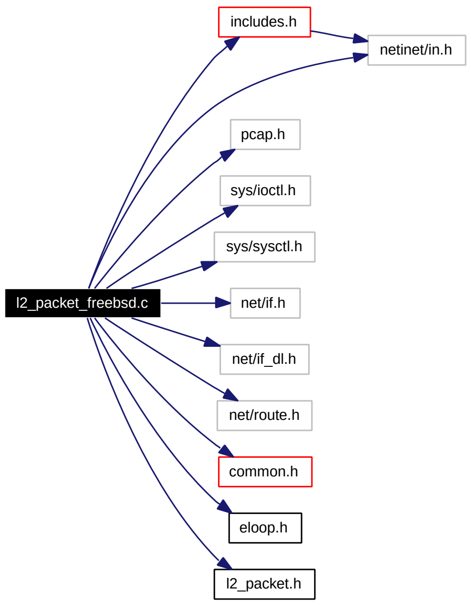
Definition at line 52 of file l2_packet_freebsd.c.

6.92 l2_packet_freebsd.c File Reference

WPA Supplicant - Layer2 packet handling with FreeBSD.

```
#include "includes.h"
#include <pcap.h>
#include <sys/ioctl.h>
#include <sys/sysctl.h>
#include <net/if.h>
#include <net/if_dl.h>
#include <net/route.h>
#include <netinet/in.h>
#include "common.h"
#include "eloop.h"
#include "l2_packet.h"
```

Include dependency graph for l2_packet_freebsd.c:



Functions

- int l2_packet_get_own_addr (struct l2_packet_data ∗l2, u8 ∗addr)

  Get own layer 2 address.

- int l2_packet_send (struct l2_packet_data ∗l2, const u8 ∗dst_addr, u16 proto, const u8 ∗buf, size_t len)

  Send a packet.

- l2_packet_data ∗ l2_packet_init (const char ∗ifname, const u8 ∗own_addr, unsigned short protocol, void(∗rx_callback)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len), void ∗rx_callback_ctx, int l2_hdr)

  Initialize l2_packet interface.

- void l2_packet_deinit (struct l2_packet_data ∗l2)

  Deinitialize l2_packet interface.

- int l2_packet_get_ip_addr (struct l2_packet_data ∗l2, char ∗buf, size_t len)

  Get the current IP address from the interface.

- void l2_packet_notify_auth_start (struct l2_packet_data ∗l2)

  Notify l2_packet about start of authentication.

6.92.1 Detailed Description

WPA Supplicant - Layer2 packet handling with FreeBSD.

Copyright

Copyright (c) 2003-2005, Jouni Malinen <jkmaline@cc.hut.fi> Copyright (c) 2005, Sam Leffler <sam@errno.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file l2_packet_freebsd.c.

6.92.2 Function Documentation

6.92.2.1 void l2_packet_deinit (struct l2_packet_data ∗ *l2*)

Deinitialize l2_packet interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

Definition at line 225 of file l2_packet_freebsd.c.

6.92.2.2 int l2_packet_get_ip_addr (struct l2_packet_data ∗ *l2*, char ∗ *buf*, size_t *len*)

Get the current IP address from the interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

buf Buffer for the IP address in text format

len Maximum buffer length

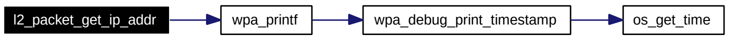
Returns:

0 on success, -1 on failure

This function can be used to get the current IP address from the interface bound to the l2_packet. This is mainly for status information and the IP address will be stored as an ASCII string. This function is not essential for wpa_supplicant operation, so full implementation is not required. l2_packet implementation will need to define the function, but it can return -1 if the IP address information is not available.

Definition at line 235 of file l2_packet_freebsd.c.

Here is the call graph for this function:



6.92.2.3 int l2_packet_get_own_addr (struct l2_packet_data ∗ *l2*, u8 ∗ *addr*)

Get own layer 2 address.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

addr Buffer for the own address (6 bytes)

Returns:

0 on success, -1 on failure

Definition at line 45 of file l2_packet_freebsd.c.

6.92.2.4 struct l2_packet_data∗ l2_packet_init (const char ∗ *ifname*, const u8 ∗ *own_addr*, unsigned short *protocol*, void(∗)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len) *rx_callback*, void ∗ *rx_callback_ctx*, int *l2_hdr*)

Initialize l2_packet interface.

Parameters:

ifname Interface name

own_addr Optional own MAC address if available from driver interface or NULL if not available

protocol Ethernet protocol number in host byte order

rx_callback Callback function that will be called for each received packet

rx_callback_ctx Callback data (ctx) for calls to rx_callback()

l2_hdr 1 = include layer 2 header, 0 = do not include header

Returns:

Pointer to internal data or NULL on failure

rx_callback function will be called with src_addr pointing to the source address (MAC address) of the the packet. If l2_hdr is set to 0, buf points to len bytes of the payload after the layer 2 header and similarly, TX buffers start with payload. This behavior can be changed by setting l2_hdr=1 to include the layer 2 header in the data buffer.

Definition at line 193 of file l2_packet_freebsd.c.

Here is the call graph for this function:

6.92.2.5 void l2_packet_notify_auth_start (struct l2_packet_data ∗ *l2*)

Notify l2_packet about start of authentication.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

This function is called when authentication is expected to start, e.g., when association has been completed, in order to prepare l2_packet implementation for EAPOL frames. This function is used mainly if the l2_-packet code needs to do polling in which case it can increasing polling frequency. This can also be an empty function if the l2_packet implementation does not benefit from knowing about the starting authentication.

Definition at line 271 of file l2_packet_freebsd.c.

6.92.2.6 int l2_packet_send (struct l2_packet_data ∗ *l2*, const u8 ∗ *dst_addr*, u16 *proto*, const u8 ∗ *buf*, size_t *len*)

Send a packet.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

dst_addr Destination address for the packet (only used if l2_hdr == 0)

proto Protocol/ethertype for the packet in host byte order (only used if l2_hdr == 0)

buf Packet contents to be sent; including layer 2 header if l2_hdr was set to 1 in l2_packet_init() call. Otherwise, only the payload of the packet is included.

len Length of the buffer (including l2 header only if l2_hdr == 1)

Returns:

>=0 on success, <0 on failure

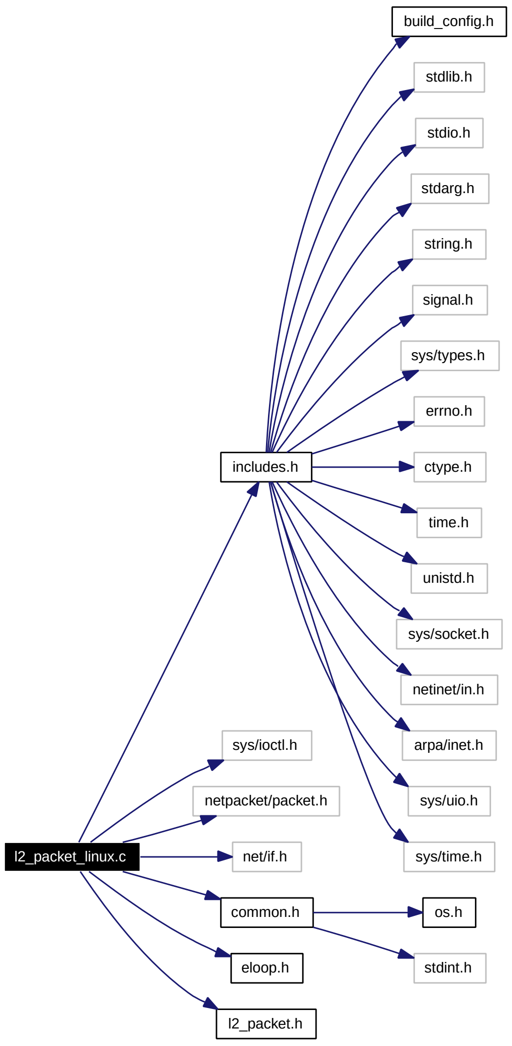
Definition at line 52 of file l2_packet_freebsd.c.

6.93 l2_packet_linux.c File Reference

WPA Supplicant - Layer2 packet handling with Linux packet sockets.

```
#include "includes.h"
#include <sys/ioctl.h>
#include <netpacket/packet.h>
#include <net/if.h>
#include "common.h"
#include "eloop.h"
#include "l2_packet.h"
```

Include dependency graph for l2_packet_linux.c:



Data Structures

- struct **l2_packet_data**

Functions

- int l2_packet_get_own_addr (struct l2_packet_data ∗l2, u8 ∗addr)

  Get own layer 2 address.

- int l2_packet_send (struct l2_packet_data ∗l2, const u8 ∗dst_addr, u16 proto, const u8 ∗buf, size_t len)

  Send a packet.

- l2_packet_data ∗ l2_packet_init (const char ∗ifname, const u8 ∗own_addr, unsigned short protocol, void(∗rx_callback)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len), void ∗rx_callback_ctx, int l2_hdr)

  Initialize l2_packet interface.

- void l2_packet_deinit (struct l2_packet_data ∗l2)

  Deinitialize l2_packet interface.

- int l2_packet_get_ip_addr (struct l2_packet_data ∗l2, char ∗buf, size_t len)

  Get the current IP address from the interface.

- void l2_packet_notify_auth_start (struct l2_packet_data ∗l2)

  Notify l2_packet about start of authentication.

6.93.1 Detailed Description

WPA Supplicant - Layer2 packet handling with Linux packet sockets.

Copyright

Copyright (c) 2003-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file l2_packet_linux.c.

6.93.2 Function Documentation

6.93.2.1 void l2_packet_deinit (struct l2_packet_data ∗ *l2*)

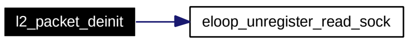
Deinitialize l2_packet interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

Definition at line 153 of file l2_packet_linux.c.

Here is the call graph for this function:



6.93.2.2 int l2_packet_get_ip_addr (struct l2_packet_data ∗ *l2*, char ∗ *buf*, size_t *len*)

Get the current IP address from the interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

buf Buffer for the IP address in text format

len Maximum buffer length

Returns:

0 on success, -1 on failure

This function can be used to get the current IP address from the interface bound to the l2_packet. This is mainly for status information and the IP address will be stored as an ASCII string. This function is not essential for wpa_supplicant operation, so full implementation is not required. l2_packet implementation will need to define the function, but it can return -1 if the IP address information is not available.

Definition at line 167 of file l2_packet_linux.c.

6.93.2.3 int l2_packet_get_own_addr (struct l2_packet_data ∗ *l2*, u8 ∗ *addr*)

Get own layer 2 address.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

addr Buffer for the own address (6 bytes)

Returns:

0 on success, -1 on failure

Definition at line 39 of file l2_packet_linux.c.

6.93.2.4 struct l2_packet_data∗ l2_packet_init (const char ∗ *ifname*, const u8 ∗ *own_addr*, unsigned short *protocol*, void(∗)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len) *rx_callback*, void ∗ *rx_callback_ctx*, int *l2_hdr*)

Initialize l2_packet interface.

Parameters:

ifname Interface name

own_addr Optional own MAC address if available from driver interface or NULL if not available

protocol Ethernet protocol number in host byte order

rx_callback Callback function that will be called for each received packet

rx_callback_ctx Callback data (ctx) for calls to rx_callback()

l2_hdr 1 = include layer 2 header, 0 = do not include header

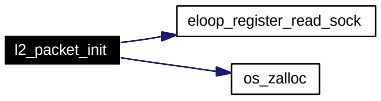
Returns:

Pointer to internal data or NULL on failure

rx_callback function will be called with src_addr pointing to the source address (MAC address) of the the packet. If l2_hdr is set to 0, buf points to len bytes of the payload after the layer 2 header and similarly, TX buffers start with payload. This behavior can be changed by setting l2_hdr=1 to include the layer 2 header in the data buffer.

Definition at line 94 of file l2_packet_linux.c.

Here is the call graph for this function:



6.93.2.5 void l2_packet_notify_auth_start (struct l2_packet_data * *l2*)

Notify l2_packet about start of authentication.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

This function is called when authentication is expected to start, e.g., when association has been completed, in order to prepare l2_packet implementation for EAPOL frames. This function is used mainly if the l2_-packet code needs to do polling in which case it can increasing polling frequency. This can also be an empty function if the l2_packet implementation does not benefit from knowing about the starting authentication.

Definition at line 194 of file l2_packet_linux.c.

6.93.2.6 int l2_packet_send (struct l2_packet_data * *l2*, const u8 * *dst_addr*, u16 *proto*, const u8 * *buf*, size_t *len*)

Send a packet.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

dst_addr Destination address for the packet (only used if l2_hdr == 0)

proto Protocol/ethertype for the packet in host byte order (only used if l2_hdr == 0)

buf Packet contents to be sent; including layer 2 header if l2_hdr was set to 1 in l2_packet_init() call. Otherwise, only the payload of the packet is included.

len Length of the buffer (including l2 header only if l2_hdr == 1)

Returns:

>=0 on success, <0 on failure

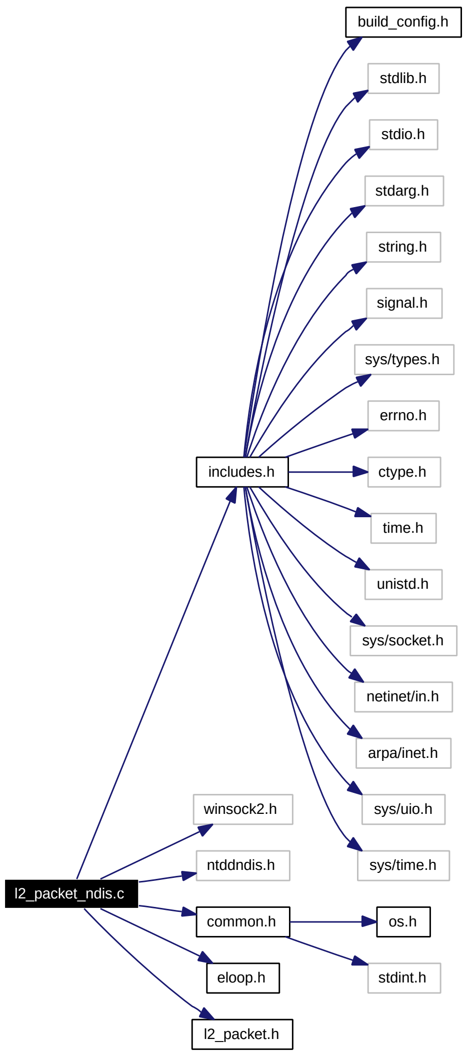
Definition at line 46 of file l2_packet_linux.c.

6.94 l2_packet_ndis.c File Reference

WPA Supplicant - Layer2 packet handling with Microsoft NDISUIO.

```
#include "includes.h"
#include <winsock2.h>
#include <ntddndis.h>
#include "common.h"
#include "eloop.h"
#include "l2_packet.h"
```

Include dependency graph for l2_packet_ndis.c:



Data Structures

- struct **l2_packet_data**

Defines

- #define **FSCTL_NDISUIO_BASE** FILE_DEVICE_NETWORK
- #define **_NDISUIO_CTL_CODE**(_Function, _Method, _Access) CTL_CODE(FSCTL_-NDISUIO_BASE, _Function, _Method, _Access)
- #define **IOCTL_NDISUIO_SET_ETHER_TYPE**

Functions

- HANDLE **driver_ndis_get_ndisuio_handle** (void)
- int l2_packet_get_own_addr (struct l2_packet_data ∗l2, u8 ∗addr)

  Get own layer 2 address.
- int l2_packet_send (struct l2_packet_data ∗l2, const u8 ∗dst_addr, u16 proto, const u8 ∗buf, size_t len)

  Send a packet.
- l2_packet_data ∗ l2_packet_init (const char ∗ifname, const u8 ∗own_addr, unsigned short protocol, void(∗rx_callback)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len), void ∗rx_callback_ctx, int l2_hdr)

  Initialize l2_packet interface.
- void l2_packet_deinit (struct l2_packet_data ∗l2)

  Deinitialize l2_packet interface.
- int l2_packet_get_ip_addr (struct l2_packet_data ∗l2, char ∗buf, size_t len)

  Get the current IP address from the interface.
- void l2_packet_notify_auth_start (struct l2_packet_data ∗l2)

  Notify l2_packet about start of authentication.

6.94.1 Detailed Description

WPA Supplicant - Layer2 packet handling with Microsoft NDISUIO.

This implementation requires Windows specific event loop implementation, i.e., eloop_win.c. In addition, the NDISUIO connection is shared with driver_ndis.c, so only that driver interface can be used and CONFIG_USE_NDISUIO must be defined.

WinXP version of the code uses overlapped I/O and a single threaded design with callback functions from I/O code. WinCE version uses a separate RX thread that blocks on ReadFile() whenever the media status is connected.

Definition in file l2_packet_ndis.c.

6.94.2 Define Documentation

6.94.2.1 #define IOCTL_NDISUIO_SET_ETHER_TYPE

Value:

```
_NDISUIO_CTL_CODE(0x202, METHOD_BUFFERED, \
                          FILE_READ_ACCESS | FILE_WRITE_ACCESS)
```

Definition at line 43 of file l2_packet_ndis.c.

6.94.3 Function Documentation

6.94.3.1 void l2_packet_deinit (struct l2_packet_data ∗ *l2*)

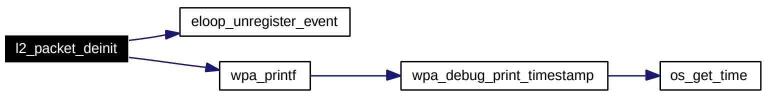
Deinitialize l2_packet interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

Definition at line 454 of file l2_packet_ndis.c.

Here is the call graph for this function:



6.94.3.2 int l2_packet_get_ip_addr (struct l2_packet_data ∗ *l2*, char ∗ *buf*, size_t *len*)

Get the current IP address from the interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

buf Buffer for the IP address in text format

len Maximum buffer length

Returns:

0 on success, -1 on failure

This function can be used to get the current IP address from the interface bound to the l2_packet. This is mainly for status information and the IP address will be stored as an ASCII string. This function is not essential for wpa_supplicant operation, so full implementation is not required. l2_packet implementation will need to define the function, but it can return -1 if the IP address information is not available.

Definition at line 509 of file l2_packet_ndis.c.

6.94.3.3 int l2_packet_get_own_addr (struct l2_packet_data * *l2*, u8 * *addr*)

Get own layer 2 address.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

addr Buffer for the own address (6 bytes)

Returns:

0 on success, -1 on failure

Definition at line 91 of file l2_packet_ndis.c.

6.94.3.4 struct l2_packet_data* l2_packet_init (const char * *ifname*, const u8 * *own_addr*, unsigned short *protocol*, void(*)(void *ctx, const u8 *src_addr, const u8 *buf, size_t len) *rx_callback*, void * *rx_callback_ctx*, int *l2_hdr*)

Initialize l2_packet interface.

Parameters:

ifname Interface name

own_addr Optional own MAC address if available from driver interface or NULL if not available

protocol Ethernet protocol number in host byte order

rx_callback Callback function that will be called for each received packet

rx_callback_ctx Callback data (ctx) for calls to rx_callback()

l2_hdr 1 = include layer 2 header, 0 = do not include header

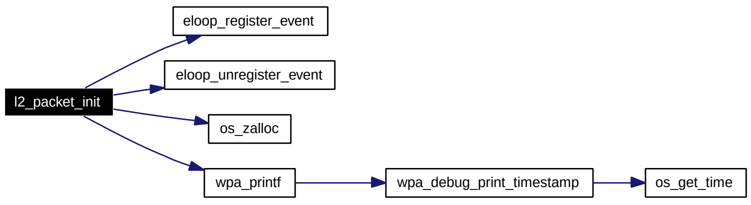
Returns:

Pointer to internal data or NULL on failure

rx_callback function will be called with src_addr pointing to the source address (MAC address) of the the packet. If l2_hdr is set to 0, buf points to len bytes of the payload after the layer 2 header and similarly, TX buffers start with payload. This behavior can be changed by setting l2_hdr=1 to include the layer 2 header in the data buffer.

Definition at line 349 of file l2_packet_ndis.c.

Here is the call graph for this function:



6.94.3.5 void l2_packet_notify_auth_start (struct l2_packet_data ∗ *l2*)

Notify l2_packet about start of authentication.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

This function is called when authentication is expected to start, e.g., when association has been completed, in order to prepare l2_packet implementation for EAPOL frames. This function is used mainly if the l2_-packet code needs to do polling in which case it can increasing polling frequency. This can also be an empty function if the l2_packet implementation does not benefit from knowing about the starting authentication.

Definition at line 515 of file l2_packet_ndis.c.

6.94.3.6 int l2_packet_send (struct l2_packet_data ∗ *l2*, const u8 ∗ *dst_addr*, u16 *proto*, const u8 ∗ *buf*, size_t *len*)

Send a packet.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

dst_addr Destination address for the packet (only used if l2_hdr == 0)

proto Protocol/ethertype for the packet in host byte order (only used if l2_hdr == 0)

buf Packet contents to be sent; including layer 2 header if l2_hdr was set to 1 in l2_packet_init() call. Otherwise, only the payload of the packet is included.

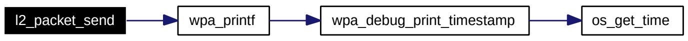
len Length of the buffer (including l2 header only if l2_hdr == 1)

Returns:

>=0 on success, <0 on failure

Definition at line 98 of file l2_packet_ndis.c.

Here is the call graph for this function:

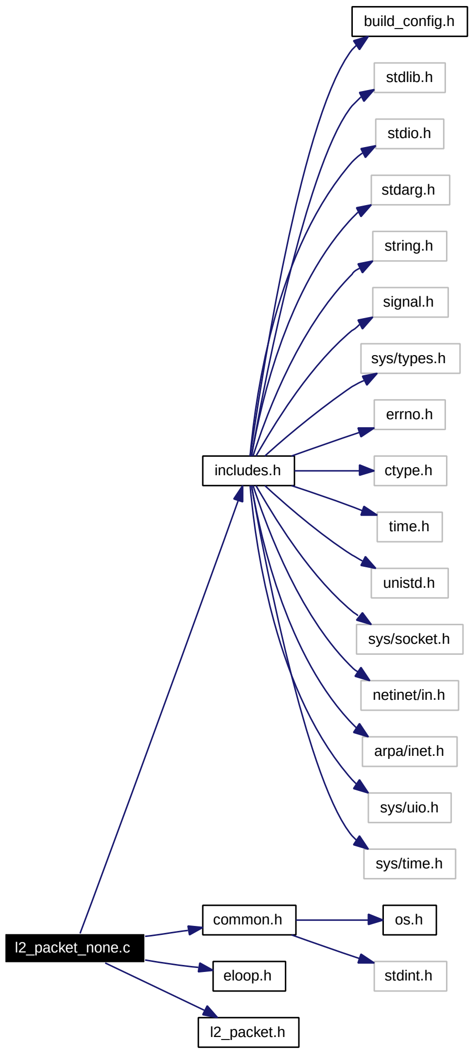


6.95 l2_packet_none.c File Reference

WPA Supplicant - Layer2 packet handling example with dummy functions.

```
#include "includes.h"
#include "common.h"
#include "eloop.h"
#include "l2_packet.h"
```

Include dependency graph for l2_packet_none.c:



Data Structures

- struct **l2_packet_data**

Functions

- int l2_packet_get_own_addr (struct l2_packet_data ∗l2, u8 ∗addr)

  Get own layer 2 address.

- int l2_packet_send (struct l2_packet_data ∗l2, const u8 ∗dst_addr, u16 proto, const u8 ∗buf, size_t len)

  Send a packet.

- l2_packet_data ∗ l2_packet_init (const char ∗ifname, const u8 ∗own_addr, unsigned short protocol, void(∗rx_callback)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len), void ∗rx_callback_ctx, int l2_hdr)

  Initialize l2_packet interface.

- void l2_packet_deinit (struct l2_packet_data ∗l2)

  Deinitialize l2_packet interface.

- int l2_packet_get_ip_addr (struct l2_packet_data ∗l2, char ∗buf, size_t len)

  Get the current IP address from the interface.

- void l2_packet_notify_auth_start (struct l2_packet_data ∗l2)

  Notify l2_packet about start of authentication.

6.95.1 Detailed Description

WPA Supplicant - Layer2 packet handling example with dummy functions.

Copyright
Copyright (c) 2003-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

This file can be used as a starting point for layer2 packet implementation.

Definition in file l2_packet_none.c.

6.95.2 Function Documentation

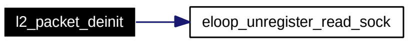
6.95.2.1 void l2_packet_deinit (struct l2_packet_data ∗ *l2*)

Deinitialize l2_packet interface.

Parameters:
l2 Pointer to internal l2_packet data from l2_packet_init()

Definition at line 100 of file l2_packet_none.c.

Here is the call graph for this function:



6.95.2.2 int l2_packet_get_ip_addr (struct l2_packet_data ∗ *l2*, char ∗ *buf*, size_t *len*)

Get the current IP address from the interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

buf Buffer for the IP address in text format

len Maximum buffer length

Returns:

0 on success, -1 on failure

This function can be used to get the current IP address from the interface bound to the l2_packet. This is mainly for status information and the IP address will be stored as an ASCII string. This function is not essential for wpa_supplicant operation, so full implementation is not required. l2_packet implementation will need to define the function, but it can return -1 if the IP address information is not available.

Definition at line 114 of file l2_packet_none.c.

6.95.2.3 int l2_packet_get_own_addr (struct l2_packet_data ∗ *l2*, u8 ∗ *addr*)

Get own layer 2 address.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

addr Buffer for the own address (6 bytes)

Returns:

0 on success, -1 on failure

Definition at line 37 of file l2_packet_none.c.

6.95.2.4 struct l2_packet_data∗ l2_packet_init (const char ∗ *ifname*, const u8 ∗ *own_addr*, unsigned short *protocol*, void(∗)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len) *rx_callback*, void ∗ *rx_callback_ctx*, int *l2_hdr*)

Initialize l2_packet interface.

Parameters:

ifname Interface name

own_addr Optional own MAC address if available from driver interface or NULL if not available

protocol Ethernet protocol number in host byte order

rx_callback Callback function that will be called for each received packet

rx_callback_ctx Callback data (ctx) for calls to rx_callback()

l2_hdr 1 = include layer 2 header, 0 = do not include header

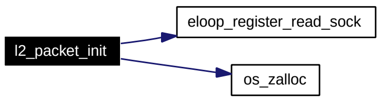
Returns:

Pointer to internal data or NULL on failure

rx_callback function will be called with src_addr pointing to the source address (MAC address) of the the packet. If l2_hdr is set to 0, buf points to len bytes of the payload after the layer 2 header and similarly, TX buffers start with payload. This behavior can be changed by setting l2_hdr=1 to include the layer 2 header in the data buffer.

Definition at line 74 of file l2_packet_none.c.

Here is the call graph for this function:



6.95.2.5 void l2_packet_notify_auth_start (struct l2_packet_data ∗ *l2*)

Notify l2_packet about start of authentication.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

This function is called when authentication is expected to start, e.g., when association has been completed, in order to prepare l2_packet implementation for EAPOL frames. This function is used mainly if the l2_-packet code needs to do polling in which case it can increasing polling frequency. This can also be an empty function if the l2_packet implementation does not benefit from knowing about the starting authentication.

Definition at line 121 of file l2_packet_none.c.

6.95.2.6 int l2_packet_send (struct l2_packet_data ∗ *l2*, const u8 ∗ *dst_addr*, u16 *proto*, const u8 ∗ *buf*, size_t *len*)

Send a packet.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

dst_addr Destination address for the packet (only used if l2_hdr == 0)

proto Protocol/ethertype for the packet in host byte order (only used if l2_hdr == 0)

buf Packet contents to be sent; including layer 2 header if l2_hdr was set to 1 in l2_packet_init() call. Otherwise, only the payload of the packet is included.

len Length of the buffer (including l2 header only if l2_hdr == 1)

Returns:

>=0 on success, <0 on failure

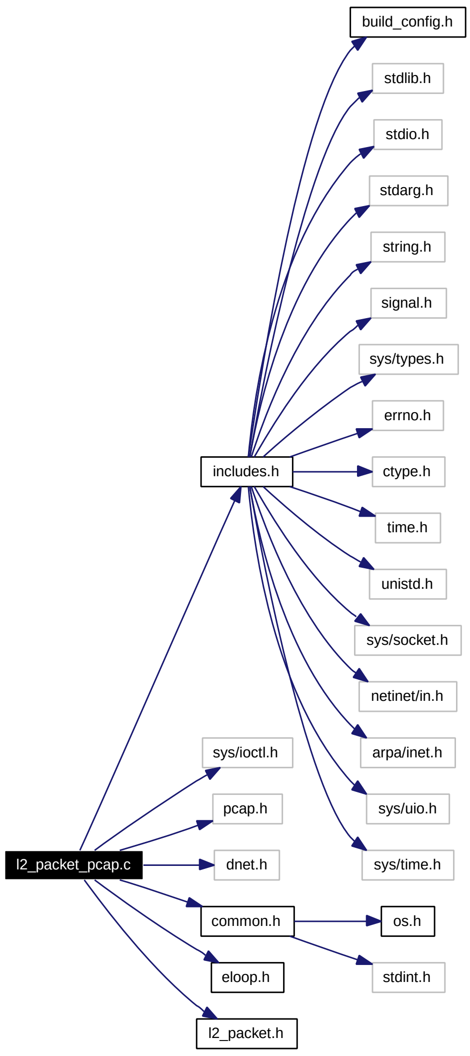
Definition at line 44 of file l2_packet_none.c.

6.96 l2_packet_pcap.c File Reference

WPA Supplicant - Layer2 packet handling with libpcap/libdnet and WinPcap.

```
#include "includes.h"
#include <sys/ioctl.h>
#include <pcap.h>
#include <dnet.h>
#include "common.h"
#include "eloop.h"
#include "l2_packet.h"
```

Include dependency graph for l2_packet_pcap.c:



Data Structures

- struct **l2_packet_data**

Functions

- int l2_packet_get_own_addr (struct l2_packet_data ∗l2, u8 ∗addr)

  Get own layer 2 address.
- int l2_packet_send (struct l2_packet_data ∗l2, const u8 ∗dst_addr, u16 proto, const u8 ∗buf, size_t len)

  Send a packet.
- l2_packet_data ∗ l2_packet_init (const char ∗ifname, const u8 ∗own_addr, unsigned short protocol, void(∗rx_callback)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len), void ∗rx_callback_ctx, int l2_hdr)

  Initialize l2_packet interface.
- void l2_packet_deinit (struct l2_packet_data ∗l2)

  Deinitialize l2_packet interface.
- int l2_packet_get_ip_addr (struct l2_packet_data ∗l2, char ∗buf, size_t len)

  Get the current IP address from the interface.
- void l2_packet_notify_auth_start (struct l2_packet_data ∗l2)

  Notify l2_packet about start of authentication.

6.96.1 Detailed Description

WPA Supplicant - Layer2 packet handling with libpcap/libdnet and WinPcap.

Definition in file l2_packet_pcap.c.

6.96.2 Function Documentation

6.96.2.1 void l2_packet_deinit (struct l2_packet_data ∗ *l2*)

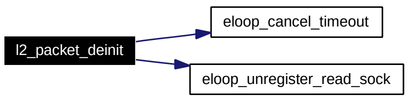
Deinitialize l2_packet interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

Definition at line 321 of file l2_packet_pcap.c.

Here is the call graph for this function:



6.96.2.2 int l2_packet_get_ip_addr (struct l2_packet_data ∗ *l2*, char ∗ *buf*, size_t *len*)

Get the current IP address from the interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

buf Buffer for the IP address in text format

len Maximum buffer length

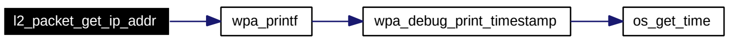
Returns:

0 on success, -1 on failure

This function can be used to get the current IP address from the interface bound to the l2_packet. This is mainly for status information and the IP address will be stored as an ASCII string. This function is not essential for wpa_supplicant operation, so full implementation is not required. l2_packet implementation will need to define the function, but it can return -1 if the IP address information is not available.

Definition at line 339 of file l2_packet_pcap.c.

Here is the call graph for this function:



6.96.2.3 int l2_packet_get_own_addr (struct l2_packet_data ∗ *l2*, u8 ∗ *addr*)

Get own layer 2 address.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

addr Buffer for the own address (6 bytes)

Returns:

0 on success, -1 on failure

Definition at line 50 of file l2_packet_pcap.c.

6.96.2.4 struct l2_packet_data∗ l2_packet_init (const char ∗ *ifname*, const u8 ∗ *own_addr*, unsigned short *protocol*, void(∗)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len) *rx_callback*, void ∗ *rx_callback_ctx*, int *l2_hdr*)

Initialize l2_packet interface.

Parameters:

ifname Interface name

own_addr Optional own MAC address if available from driver interface or NULL if not available

protocol Ethernet protocol number in host byte order

rx_callback Callback function that will be called for each received packet

rx_callback_ctx Callback data (ctx) for calls to rx_callback()

l2_hdr 1 = include layer 2 header, 0 = do not include header

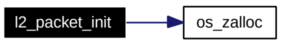
Returns:

Pointer to internal data or NULL on failure

rx_callback function will be called with src_addr pointing to the source address (MAC address) of the the packet. If l2_hdr is set to 0, buf points to len bytes of the payload after the layer 2 header and similarly, TX buffers start with payload. This behavior can be changed by setting l2_hdr=1 to include the layer 2 header in the data buffer.

Definition at line 285 of file l2_packet_pcap.c.

Here is the call graph for this function:



6.96.2.5 void l2_packet_notify_auth_start (struct l2_packet_data ∗ *l2*)

Notify l2_packet about start of authentication.

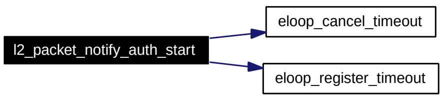
Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

This function is called when authentication is expected to start, e.g., when association has been completed, in order to prepare l2_packet implementation for EAPOL frames. This function is used mainly if the l2_-packet code needs to do polling in which case it can increasing polling frequency. This can also be an empty function if the l2_packet implementation does not benefit from knowing about the starting authentication.

Definition at line 375 of file l2_packet_pcap.c.

Here is the call graph for this function:



6.96.2.6 int l2_packet_send (struct l2_packet_data ∗ *l2*, const u8 ∗ *dst_addr*, u16 *proto*, const u8 ∗ *buf*, size_t *len*)

Send a packet.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

dst_addr Destination address for the packet (only used if l2_hdr == 0)

proto Protocol/ethertype for the packet in host byte order (only used if l2_hdr == 0)

buf Packet contents to be sent; including layer 2 header if l2_hdr was set to 1 in l2_packet_init() call. Otherwise, only the payload of the packet is included.

len Length of the buffer (including l2 header only if l2_hdr == 1)

Returns:

>=0 on success, <0 on failure

Definition at line 84 of file l2_packet_pcap.c.

6.97 l2_packet_winpcap.c File Reference

WPA Supplicant - Layer2 packet handling with WinPcap RX thread.

`#include "includes.h"`

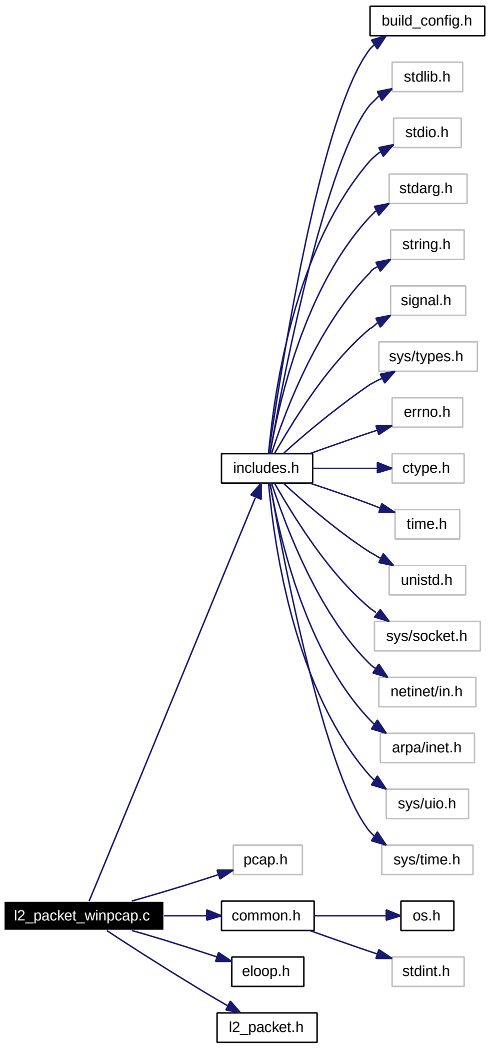
`#include <pcap.h>`

`#include "common.h"`

`#include "eloop.h"`

`#include "l2_packet.h"`

Include dependency graph for l2_packet_winpcap.c:



Data Structures

- struct **l2_packet_data**

Functions

- int l2_packet_get_own_addr (struct l2_packet_data ∗l2, u8 ∗addr)

  Get own layer 2 address.

- int l2_packet_send (struct l2_packet_data ∗l2, const u8 ∗dst_addr, u16 proto, const u8 ∗buf, size_t len)

  Send a packet.

- l2_packet_data ∗ l2_packet_init (const char ∗ifname, const u8 ∗own_addr, unsigned short protocol, void(∗rx_callback)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len), void ∗rx_callback_ctx, int l2_hdr)

  Initialize l2_packet interface.

- void l2_packet_deinit (struct l2_packet_data ∗l2)

  Deinitialize l2_packet interface.

- int l2_packet_get_ip_addr (struct l2_packet_data ∗l2, char ∗buf, size_t len)

  Get the current IP address from the interface.

- void l2_packet_notify_auth_start (struct l2_packet_data ∗l2)

  Notify l2_packet about start of authentication.

6.97.1 Detailed Description

WPA Supplicant - Layer2 packet handling with WinPcap RX thread.

Copyright

Copyright (c) 2003-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

This l2_packet implementation is explicitly for WinPcap and Windows events. l2_packet_pcap.c has support for WinPcap, but it requires polling to receive frames which means relatively long latency for EAPOL RX processing. The implementation here uses a separate thread to allow WinPcap to be receiving all the time to reduce latency for EAPOL receiving from about 100 ms to 3 ms when comparing l2_packet_pcap.c to l2_packet_winpcap.c. Extra sleep of 50 ms is added in to receive thread whenever no EAPOL frames has been received for a while. Whenever an EAPOL handshake is expected, this sleep is removed.

The RX thread receives a frame and signals main thread through Windows event about the availability of a new frame. Processing the received frame is synchronized with pair of Windows events so that no extra buffer or queuing mechanism is needed. This implementation requires Windows specific event loop implementation, i.e., eloop_win.c.

WinPcap has pcap_getevent() that could, in theory at least, be used to implement this kind of waiting with a simpler single-thread design. However, that event handle is not really signaled immediately when receiving each frame, so it does not really work for this kind of use.

Definition in file l2_packet_winpcap.c.

6.97.2 Function Documentation

6.97.2.1 void l2_packet_deinit (struct l2_packet_data ∗ *l2*)

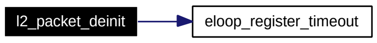
Deinitialize l2_packet interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

Definition at line 280 of file l2_packet_winpcap.c.

Here is the call graph for this function:



6.97.2.2 int l2_packet_get_ip_addr (struct l2_packet_data ∗ *l2*, char ∗ *buf*, size_t *len*)

Get the current IP address from the interface.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

buf Buffer for the IP address in text format

len Maximum buffer length

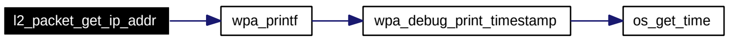
Returns:

0 on success, -1 on failure

This function can be used to get the current IP address from the interface bound to the l2_packet. This is mainly for status information and the IP address will be stored as an ASCII string. This function is not essential for wpa_supplicant operation, so full implementation is not required. l2_packet implementation will need to define the function, but it can return -1 if the IP address information is not available.

Definition at line 302 of file l2_packet_winpcap.c.

Here is the call graph for this function:



6.97.2.3 int l2_packet_get_own_addr (struct l2_packet_data ∗ *l2*, u8 ∗ *addr*)

Get own layer 2 address.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

addr Buffer for the own address (6 bytes)

Returns:

0 on success, -1 on failure

Definition at line 72 of file l2_packet_winpcap.c.

6.97.2.4 struct l2_packet_data∗ l2_packet_init (const char ∗ *ifname*, const u8 ∗ *own_addr*, unsigned short *protocol*, void(∗)(void ∗ctx, const u8 ∗src_addr, const u8 ∗buf, size_t len) *rx_callback*, void ∗ *rx_callback_ctx*, int *l2_hdr*)

Initialize l2_packet interface.

Parameters:

ifname Interface name

own_addr Optional own MAC address if available from driver interface or NULL if not available

protocol Ethernet protocol number in host byte order

rx_callback Callback function that will be called for each received packet

rx_callback_ctx Callback data (ctx) for calls to rx_callback()

l2_hdr 1 = include layer 2 header, 0 = do not include header

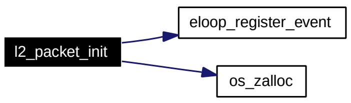
Returns:

Pointer to internal data or NULL on failure

rx_callback function will be called with src_addr pointing to the source address (MAC address) of the the packet. If l2_hdr is set to 0, buf points to len bytes of the payload after the layer 2 header and similarly, TX buffers start with payload. This behavior can be changed by setting l2_hdr=1 to include the layer 2 header in the data buffer.

Definition at line 205 of file l2_packet_winpcap.c.

Here is the call graph for this function:



6.97.2.5 void l2_packet_notify_auth_start (struct l2_packet_data ∗ *l2*)

Notify l2_packet about start of authentication.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

This function is called when authentication is expected to start, e.g., when association has been completed, in order to prepare l2_packet implementation for EAPOL frames. This function is used mainly if the l2_-packet code needs to do polling in which case it can increasing polling frequency. This can also be an empty function if the l2_packet implementation does not benefit from knowing about the starting authentication.

Definition at line 338 of file l2_packet_winpcap.c.

6.97.2.6 int l2_packet_send (struct l2_packet_data ∗ *l2*, const u8 ∗ *dst_addr*, u16 *proto*, const u8 ∗ *buf*, size_t *len*)

Send a packet.

Parameters:

l2 Pointer to internal l2_packet data from l2_packet_init()

dst_addr Destination address for the packet (only used if l2_hdr == 0)

proto Protocol/ethertype for the packet in host byte order (only used if l2_hdr == 0)

buf Packet contents to be sent; including layer 2 header if l2_hdr was set to 1 in l2_packet_init() call. Otherwise, only the payload of the packet is included.

len Length of the buffer (including l2 header only if l2_hdr == 1)

Returns:

>=0 on success, <0 on failure

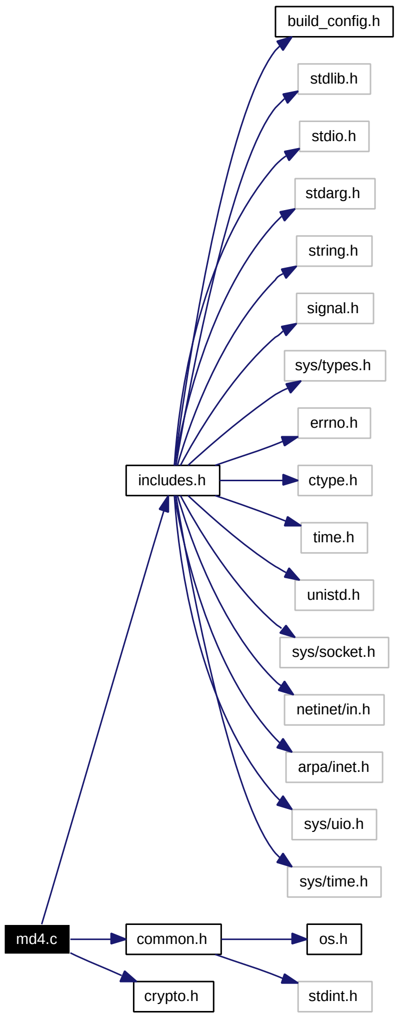
Definition at line 79 of file l2_packet_winpcap.c.

6.98 md4.c File Reference

MD4 hash implementation.

```
#include "includes.h"
#include "common.h"
#include "crypto.h"
```

Include dependency graph for md4.c:



6.98.1 Detailed Description

MD4 hash implementation.

Copyright

Copyright (c) 2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

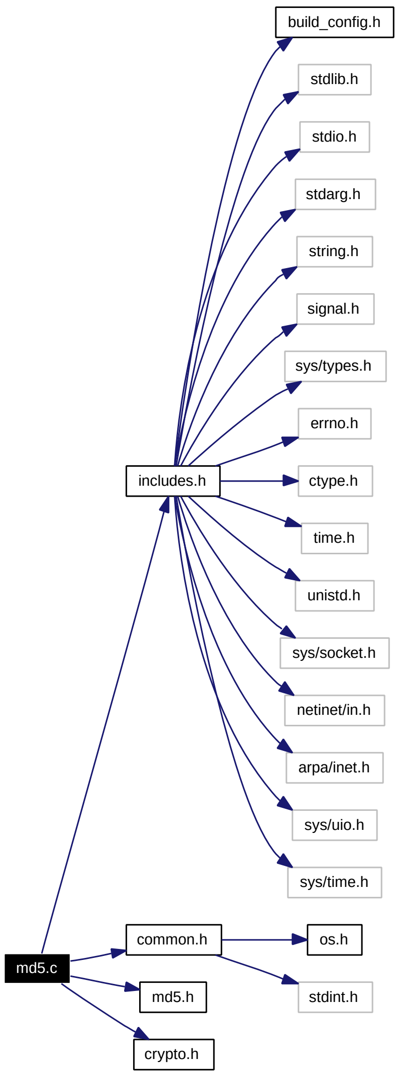
Definition in file md4.c.

6.99 md5.c File Reference

MD5 hash implementation and interface functions.

```
#include "includes.h"
#include "common.h"
#include "md5.h"
#include "crypto.h"
```

Include dependency graph for md5.c:



Functions

- void hmac_md5_vector (const u8 ∗key, size_t key_len, size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

    HMAC-MD5 over data vector (RFC 2104).

- void hmac_md5 (const u8 ∗key, size_t key_len, const u8 ∗data, size_t data_len, u8 ∗mac)
  HMAC-MD5 over data buffer (RFC 2104).

6.99.1 Detailed Description

MD5 hash implementation and interface functions.

Copyright
Copyright (c) 2003-2005, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file md5.c.

6.99.2 Function Documentation

6.99.2.1 void hmac_md5 (const u8 ∗ *key*, size_t *key_len*, const u8 ∗ *data*, size_t *data_len*, u8 ∗ *mac*)

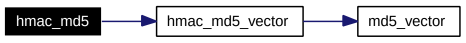
HMAC-MD5 over data buffer (RFC 2104).

Parameters:

key	Key for HMAC operations
key_len	Length of the key in bytes
data	Pointers to the data area
data_len	Length of the data area
mac	Buffer for the hash (16 bytes)

Definition at line 106 of file md5.c.

Here is the call graph for this function:



6.99.2.2 void hmac_md5_vector (const u8 ∗ *key*, size_t *key_len*, size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

HMAC-MD5 over data vector (RFC 2104).

Parameters:

key	Key for HMAC operations
key_len	Length of the key in bytes

num_elem Number of elements in the data vector

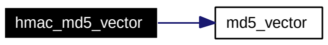
addr Pointers to the data areas

len Lengths of the data blocks

mac Buffer for the hash (16 bytes)

Definition at line 33 of file md5.c.

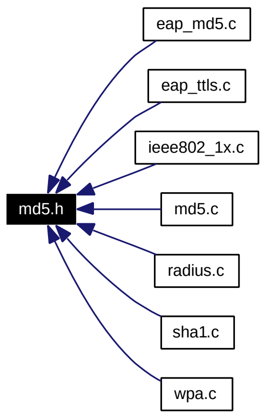
Here is the call graph for this function:



6.100 md5.h File Reference

MD5 hash implementation and interface functions.

This graph shows which files directly or indirectly include this file:



Defines

- #define **MD5_MAC_LEN** 16

Functions

- void hmac_md5_vector (const u8 ∗key, size_t key_len, size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

    HMAC-MD5 over data vector (RFC 2104).

- void hmac_md5 (const u8 ∗key, size_t key_len, const u8 ∗data, size_t data_len, u8 ∗mac)

    HMAC-MD5 over data buffer (RFC 2104).

6.100.1 Detailed Description

MD5 hash implementation and interface functions.

Copyright

Copyright (c) 2003-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file md5.h.

6.100.2 Function Documentation

6.100.2.1 void hmac_md5 (const u8 ∗ *key*, size_t *key_len*, const u8 ∗ *data*, size_t *data_len*, u8 ∗ *mac*)

HMAC-MD5 over data buffer (RFC 2104).

Parameters:

key Key for HMAC operations

key_len Length of the key in bytes

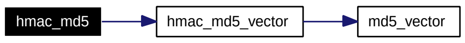
data Pointers to the data area

data_len Length of the data area

mac Buffer for the hash (16 bytes)

Definition at line 106 of file md5.c.

Here is the call graph for this function:



6.100.2.2 void hmac_md5_vector (const u8 ∗ *key*, size_t *key_len*, size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

HMAC-MD5 over data vector (RFC 2104).

Parameters:

key Key for HMAC operations

key_len Length of the key in bytes

num_elem Number of elements in the data vector

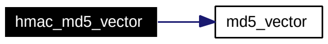
addr Pointers to the data areas

len Lengths of the data blocks

mac Buffer for the hash (16 bytes)

Definition at line 33 of file md5.c.

Here is the call graph for this function:

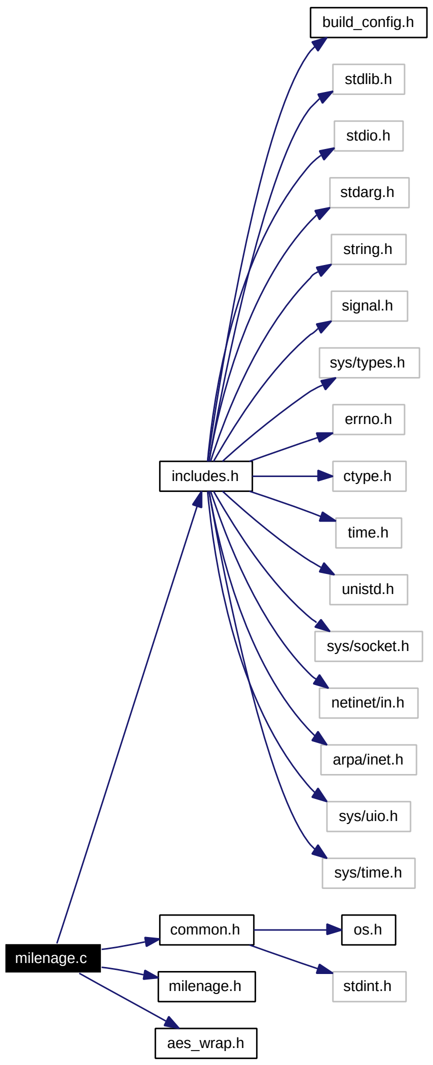


6.101 milenage.c File Reference

3GPP AKA - Milenage algorithm (3GPP TS 35.205, .206, .207, .208)

```
#include "includes.h"
#include "common.h"
#include "milenage.h"
#include "aes_wrap.h"
```

Include dependency graph for milenage.c:



Functions

- void milenage_generate (const u8 ∗opc, const u8 ∗amf, const u8 ∗k, const u8 ∗sqn, const u8 ∗_rand, u8 ∗autn, u8 ∗ik, u8 ∗ck, u8 ∗res, size_t ∗res_len)

    Generate AKA AUTN,IK,CK,RES.

- int milenage_auts (const u8 ∗opc, const u8 ∗k, const u8 ∗_rand, const u8 ∗auts, u8 ∗sqn)

  Milenage AUTS validation.

- void gsm_milenage (const u8 ∗opc, const u8 ∗k, const u8 ∗_rand, u8 ∗sres, u8 ∗kc)

  Generate GSM-Milenage (3GPP TS 55.205) authentication triplet.

6.101.1 Detailed Description

3GPP AKA - Milenage algorithm (3GPP TS 35.205, .206, .207, .208)

Copyright

Copyright (c) 2006 <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

This file implements an example authentication algorithm defined for 3GPP AKA. This can be used to implement a simple HLR/AuC into hlr_auc_gw to allow EAP-AKA to be tested properly with real USIM cards.

This implementations assumes that the r1..r5 and c1..c5 constants defined in TS 35.206 are used, i.e., r1=64, r2=0, r3=32, r4=64, r5=96, c1=00..00, c2=00..01, c3=00..02, c4=00..04, c5=00..08. The block cipher is assumed to be AES (Rijndael).

Definition in file milenage.c.

6.101.2 Function Documentation

6.101.2.1 void gsm_milenage (const u8 ∗ *opc*, const u8 ∗ *k*, const u8 ∗ *_rand*, u8 ∗ *sres*, u8 ∗ *kc*)

Generate GSM-Milenage (3GPP TS 55.205) authentication triplet.

Parameters:

opc	OPc = 128-bit operator variant algorithm configuration field (encr.)
k	K = 128-bit subscriber key
_rand	RAND = 128-bit random challenge
sres	Buffer for SRES = 32-bit SRES
kc	Buffer for Kc = 64-bit Kc

Definition at line 230 of file milenage.c.

6.101.2.2 int milenage_auts (const u8 ∗ *opc*, const u8 ∗ *k*, const u8 ∗ *_rand*, const u8 ∗ *auts*, u8 ∗ *sqn*)

Milenage AUTS validation.

Parameters:

opc OPc = 128-bit operator variant algorithm configuration field (encr.)

k K = 128-bit subscriber key

_rand RAND = 128-bit random challenge

auts AUTS = 112-bit authentication token from client

sqn Buffer for SQN = 48-bit sequence number

Returns:

0 = success (sqn filled), -1 on failure

Definition at line 204 of file milenage.c.

6.101.2.3 void milenage_generate (const u8 ∗ *opc*, const u8 ∗ *amf*, const u8 ∗ *k*, const u8 ∗ *sqn*, const u8 ∗ *_rand*, u8 ∗ *autn*, u8 ∗ *ik*, u8 ∗ *ck*, u8 ∗ *res*, size_t ∗ *res_len*)

Generate AKA AUTN,IK,CK,RES.

Parameters:

opc OPc = 128-bit operator variant algorithm configuration field (encr.)

amf AMF = 16-bit authentication management field

k K = 128-bit subscriber key

sqn SQN = 48-bit sequence number

_rand RAND = 128-bit random challenge

autn Buffer for AUTN = 128-bit authentication token

ik Buffer for IK = 128-bit integrity key (f4), or NULL

ck Buffer for CK = 128-bit confidentiality key (f3), or NULL

res Buffer for RES = 64-bit signed response (f2), or NULL

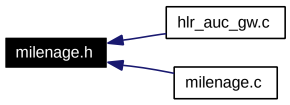
res_len Max length for res; set to used length or 0 on failure

Definition at line 171 of file milenage.c.

6.102 milenage.h File Reference

UMTS AKA - Milenage algorithm (3GPP TS 35.205, .206, .207, .208).

This graph shows which files directly or indirectly include this file:



Functions

- void milenage_generate (const u8 ∗opc, const u8 ∗amf, const u8 ∗k, const u8 ∗sqn, const u8 ∗_rand, u8 ∗autn, u8 ∗ik, u8 ∗ck, u8 ∗res, size_t ∗res_len)

  Generate AKA AUTN,IK,CK,RES.

- int milenage_auts (const u8 ∗opc, const u8 ∗k, const u8 ∗_rand, const u8 ∗auts, u8 ∗sqn)

  Milenage AUTS validation.

- void gsm_milenage (const u8 ∗opc, const u8 ∗k, const u8 ∗_rand, u8 ∗sres, u8 ∗kc)

  Generate GSM-Milenage (3GPP TS 55.205) authentication triplet.

6.102.1 Detailed Description

UMTS AKA - Milenage algorithm (3GPP TS 35.205, .206, .207, .208).

Copyright

Copyright (c) 2006 <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file milenage.h.

6.102.2 Function Documentation

6.102.2.1 void gsm_milenage (const u8 ∗ *opc*, const u8 ∗ *k*, const u8 ∗ *_rand*, u8 ∗ *sres*, u8 ∗ *kc*)

Generate GSM-Milenage (3GPP TS 55.205) authentication triplet.

Parameters:

opc OPc = 128-bit operator variant algorithm configuration field (encr.)

k K = 128-bit subscriber key

_rand RAND = 128-bit random challenge

sres Buffer for SRES = 32-bit SRES

kc Buffer for Kc = 64-bit Kc

Definition at line 230 of file milenage.c.

6.102.2.2 int milenage_auts (const u8 ∗ *opc*, const u8 ∗ *k*, const u8 ∗ *_rand*, const u8 ∗ *auts*, u8 ∗ *sqn*)

Milenage AUTS validation.

Parameters:

opc OPc = 128-bit operator variant algorithm configuration field (encr.)

k K = 128-bit subscriber key

_rand RAND = 128-bit random challenge

auts AUTS = 112-bit authentication token from client

sqn Buffer for SQN = 48-bit sequence number

Returns:

0 = success (sqn filled), -1 on failure

Definition at line 204 of file milenage.c.

6.102.2.3 void milenage_generate (const u8 ∗ *opc*, const u8 ∗ *amf*, const u8 ∗ *k*, const u8 ∗ *sqn*, const u8 ∗ *_rand*, u8 ∗ *autn*, u8 ∗ *ik*, u8 ∗ *ck*, u8 ∗ *res*, size_t ∗ *res_len*)

Generate AKA AUTN,IK,CK,RES.

Parameters:

opc OPc = 128-bit operator variant algorithm configuration field (encr.)

amf AMF = 16-bit authentication management field

k K = 128-bit subscriber key

sqn SQN = 48-bit sequence number

_rand RAND = 128-bit random challenge

autn Buffer for AUTN = 128-bit authentication token

ik Buffer for IK = 128-bit integrity key (f4), or NULL

ck Buffer for CK = 128-bit confidentiality key (f3), or NULL

res Buffer for RES = 64-bit signed response (f2), or NULL

res_len Max length for res; set to used length or 0 on failure

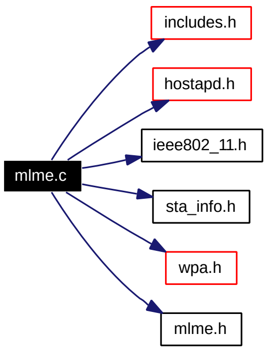
Definition at line 171 of file milenage.c.

6.103 mlme.c File Reference

hostapd / IEEE 802.11 MLME

```
#include "includes.h"
#include "hostapd.h"
#include "ieee802_11.h"
#include "sta_info.h"
#include "wpa.h"
#include "mlme.h"
```

Include dependency graph for mlme.c:



Functions

- void mlme_authenticate_indication (struct hostapd_data ∗hapd, struct sta_info ∗sta)

  Report the establishment of an authentication.

- void mlme_deauthenticate_indication (struct hostapd_data ∗hapd, struct sta_info ∗sta, u16 reason_-code)

  Report the invalidation of an.

- void mlme_associate_indication (struct hostapd_data ∗hapd, struct sta_info ∗sta)

  Report the establishment of an association with.

- void mlme_reassociate_indication (struct hostapd_data ∗hapd, struct sta_info ∗sta)

  Report the establishment of an reassociation.

- void mlme_disassociate_indication (struct hostapd_data ∗hapd, struct sta_info ∗sta, u16 reason_-code)

  Report disassociation with a specific peer.

- void **mlme_michaelmicfailure_indication** (struct hostapd_data ∗hapd, const u8 ∗addr)
- void **mlme_deletekeys_request** (struct hostapd_data ∗hapd, struct sta_info ∗sta)

6.103.1 Detailed Description

hostapd / IEEE 802.11 MLME

Copyright

Definition in file mlme.c.

6.103.2 Function Documentation

6.103.2.1 void mlme_associate_indication (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*)

Report the establishment of an association with.

a specific peer MAC entity

Parameters:

hapd	BSS data
sta	peer STA data

MLME calls this function as a result of the establishment of an association with a specific peer MAC entity that resulted from an association procedure that was initiated by that specific peer MAC entity.

PeerSTAAddress = sta->addr

Definition at line 103 of file mlme.c.

6.103.2.2 void mlme_authenticate_indication (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*)

Report the establishment of an authentication.

relationship with a specific peer MAC entity

Parameters:

hapd	BSS data
sta	peer STA data

MLME calls this function as a result of the establishment of an authentication relationship with a specific peer MAC entity that resulted from an authentication procedure that was initiated by that specific peer MAC entity.

PeerSTAAddress = sta->addr AuthenticationType = sta->auth_alg (WLAN_AUTH_OPEN / WLAN_-AUTH_SHARED_KEY)

Definition at line 55 of file mlme.c.

6.103.2.3 void mlme_deauthenticate_indication (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*, u16 *reason_code*)

Report the invalidation of an.

authentication relationship with a specific peer MAC entity

Parameters:

hapd	BSS data
sta	Peer STA data
reason_code	ReasonCode from Deauthentication frame

MLME calls this function as a result of the invalidation of an authentication relationship with a specific peer MAC entity.

PeerSTAAddress = sta->addr

Definition at line 79 of file mlme.c.

6.103.2.4 void mlme_disassociate_indication (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*, u16 *reason_code*)

Report disassociation with a specific peer.

MAC entity

Parameters:

hapd	BSS data
sta	Peer STA data
reason_code	ReasonCode from Disassociation frame

MLME calls this function as a result of the invalidation of an association relationship with a specific peer MAC entity.

PeerSTAAddress = sta->addr

Definition at line 152 of file mlme.c.

6.103.2.5 void mlme_reassociate_indication (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*)

Report the establishment of an reassociation.

with a specific peer MAC entity

Parameters:

hapd	BSS data
sta	peer STA data

MLME calls this function as a result of the establishment of an reassociation with a specific peer MAC entity that resulted from a reassociation procedure that was initiated by that specific peer MAC entity.

PeerSTAAddress = sta->addr

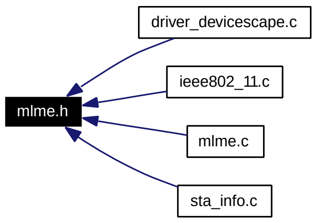
sta->previous_ap contains the "Current AP" information from ReassocReq.

Definition at line 128 of file mlme.c.

6.104 mlme.h File Reference

hostapd / IEEE 802.11 MLME

This graph shows which files directly or indirectly include this file:



Functions

- void mlme_authenticate_indication (struct hostapd_data ∗hapd, struct sta_info ∗sta)

  Report the establishment of an authentication.

- void mlme_deauthenticate_indication (struct hostapd_data ∗hapd, struct sta_info ∗sta, u16 reason_-code)

  Report the invalidation of an.

- void mlme_associate_indication (struct hostapd_data ∗hapd, struct sta_info ∗sta)

  Report the establishment of an association with.

- void mlme_reassociate_indication (struct hostapd_data ∗hapd, struct sta_info ∗sta)

  Report the establishment of an reassociation.

- void mlme_disassociate_indication (struct hostapd_data ∗hapd, struct sta_info ∗sta, u16 reason_-code)

  Report disassociation with a specific peer.

- void **mlme_michaelmicfailure_indication** (struct hostapd_data ∗hapd, const u8 ∗addr)
- void **mlme_deletekeys_request** (struct hostapd_data ∗hapd, struct sta_info ∗sta)

6.104.1 Detailed Description

hostapd / IEEE 802.11 MLME

Copyright

Definition in file mlme.h.

6.104.2 Function Documentation

6.104.2.1 void mlme_associate_indication (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*)

Report the establishment of an association with.

a specific peer MAC entity

Parameters:

hapd BSS data

sta peer STA data

MLME calls this function as a result of the establishment of an association with a specific peer MAC entity that resulted from an association procedure that was initiated by that specific peer MAC entity.

PeerSTAAddress = sta->addr

Definition at line 103 of file mlme.c.

6.104.2.2 void mlme_authenticate_indication (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*)

Report the establishment of an authentication.

relationship with a specific peer MAC entity

Parameters:

hapd BSS data

sta peer STA data

MLME calls this function as a result of the establishment of an authentication relationship with a specific peer MAC entity that resulted from an authentication procedure that was initiated by that specific peer MAC entity.

PeerSTAAddress = sta->addr AuthenticationType = sta->auth_alg (WLAN_AUTH_OPEN / WLAN_-AUTH_SHARED_KEY)

Definition at line 55 of file mlme.c.

6.104.2.3 void mlme_deauthenticate_indication (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*, u16 *reason_code*)

Report the invalidation of an.

authentication relationship with a specific peer MAC entity

Parameters:

hapd BSS data

sta Peer STA data

reason_code ReasonCode from Deauthentication frame

MLME calls this function as a result of the invalidation of an authentication relationship with a specific peer MAC entity.

PeerSTAAddress = sta->addr

Definition at line 79 of file mlme.c.

6.104.2.4 void mlme_disassociate_indication (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*, u16 *reason_code*)

Report disassociation with a specific peer.

MAC entity

Parameters:

hapd BSS data

sta Peer STA data

reason_code ReasonCode from Disassociation frame

MLME calls this function as a result of the invalidation of an association relationship with a specific peer MAC entity.

PeerSTAAddress = sta->addr

Definition at line 152 of file mlme.c.

6.104.2.5 void mlme_reassociate_indication (struct hostapd_data ∗ *hapd*, struct sta_info ∗ *sta*)

Report the establishment of an reassociation.

with a specific peer MAC entity

Parameters:

hapd BSS data

sta peer STA data

MLME calls this function as a result of the establishment of an reassociation with a specific peer MAC entity that resulted from a reassociation procedure that was initiated by that specific peer MAC entity.

PeerSTAAddress = sta->addr

sta->previous_ap contains the "Current AP" information from ReassocReq.

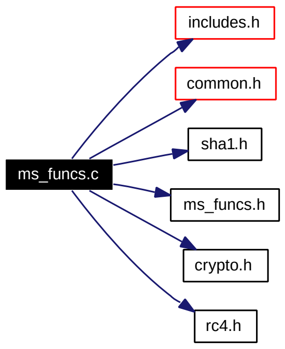
Definition at line 128 of file mlme.c.

6.105 ms_funcs.c File Reference

WPA Supplicant / shared MSCHAPV2 helper functions / RFC 2433 / RFC 2759.

```
#include "includes.h"
#include "common.h"
#include "sha1.h"
#include "ms_funcs.h"
#include "crypto.h"
#include "rc4.h"
```

Include dependency graph for ms_funcs.c:



Defines

- #define **PWBLOCK_LEN** 516

Functions

- void nt_password_hash (const u8 ∗password, size_t password_len, u8 ∗password_hash)

    NtPasswordHash() - RFC 2759, Sect. 8.3.

- void hash_nt_password_hash (const u8 ∗password_hash, u8 ∗password_hash_hash)

    HashNtPasswordHash() - RFC 2759, Sect. 8.4.

- void challenge_response (const u8 ∗challenge, const u8 ∗password_hash, u8 ∗response)

    ChallengeResponse() - RFC 2759, Sect. 8.5.

- void generate_nt_response (const u8 ∗auth_challenge, const u8 ∗peer_challenge, const u8 ∗username, size_t username_len, const u8 ∗password, size_t password_len, u8 ∗response)

    GenerateNTResponse() - RFC 2759, Sect. 8.1.

- void generate_nt_response_pwhash (const u8 ∗auth_challenge, const u8 ∗peer_challenge, const u8 ∗username, size_t username_len, const u8 ∗password_hash, u8 ∗response)

    GenerateNTResponse() - RFC 2759, Sect. 8.1.

- void generate_authenticator_response_pwhash (const u8 ∗password_hash, const u8 ∗peer_-challenge, const u8 ∗auth_challenge, const u8 ∗username, size_t username_len, const u8 ∗nt_-response, u8 ∗response)

  GenerateAuthenticatorResponse() - RFC 2759, Sect. 8.7.

- void generate_authenticator_response (const u8 ∗password, size_t password_len, const u8 ∗peer_-challenge, const u8 ∗auth_challenge, const u8 ∗username, size_t username_len, const u8 ∗nt_-response, u8 ∗response)

  GenerateAuthenticatorResponse() - RFC 2759, Sect. 8.7.

- void nt_challenge_response (const u8 ∗challenge, const u8 ∗password, size_t password_len, u8 ∗response)

  NtChallengeResponse() - RFC 2433, Sect. A.5.

- void get_master_key (const u8 ∗password_hash_hash, const u8 ∗nt_response, u8 ∗master_key)

  GetMasterKey() - RFC 3079, Sect. 3.4.

- void get_asymetric_start_key (const u8 ∗master_key, u8 ∗session_key, size_t session_key_len, int is_send, int is_server)

  GetAsymetricStartKey() - RFC 3079, Sect. 3.4.

- void new_password_encrypted_with_old_nt_password_hash (const u8 ∗new_password, size_-t new_password_len, const u8 ∗old_password, size_t old_password_len, u8 ∗encrypted_pw_block)

  NewPasswordEncryptedWithOldNtPasswordHash() - RFC 2759, Sect. 8.9.

- void old_nt_password_hash_encrypted_with_new_nt_password_hash (const u8 ∗new_password, size_t new_password_len, const u8 ∗old_password, size_t old_password_len, u8 ∗encrypted_-password_hash)

  OldNtPasswordHashEncryptedWithNewNtPasswordHash() - RFC 2759, Sect. 8.12.

6.105.1 Detailed Description

WPA Supplicant / shared MSCHAPV2 helper functions / RFC 2433 / RFC 2759.

Copyright

Definition in file ms_funcs.c.

6.105.2 Function Documentation

6.105.2.1 void challenge_response (const u8 ∗ *challenge*, const u8 ∗ *password_hash*, u8 ∗ *response*)

ChallengeResponse() - RFC 2759, Sect. 8.5.

Parameters:

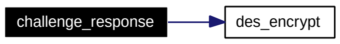
challenge 8-octet Challenge (IN)

password_hash 16-octet PasswordHash (IN)

response 24-octet Response (OUT)

Definition at line 102 of file ms_funcs.c.

Here is the call graph for this function:



6.105.2.2 void generate_authenticator_response (const u8 ∗ *password*, size_t *password_len*, const u8 ∗ *peer_challenge*, const u8 ∗ *auth_challenge*, const u8 ∗ *username*, size_t *username_len*, const u8 ∗ *nt_response*, u8 ∗ *response*)

GenerateAuthenticatorResponse() - RFC 2759, Sect. 8.7.

Parameters:

password 0-to-256-unicode-char Password (IN; ASCII)

password_len Length of password

nt_response 24-octet NT-Response (IN)

peer_challenge 16-octet PeerChallenge (IN)

auth_challenge 16-octet AuthenticatorChallenge (IN)

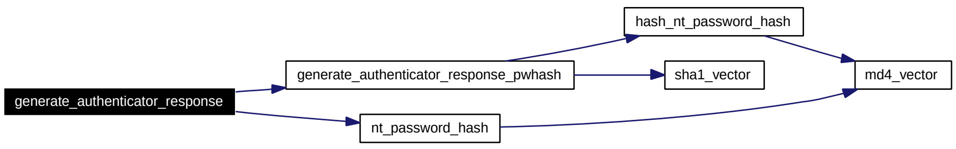
username 0-to-256-char UserName (IN)

username_len Length of username

response 20-octet AuthenticatorResponse (OUT) (note: this value is usually encoded as a 42-octet ASCII string (S=<hexdump of="" response="">)

Definition at line 233 of file ms_funcs.c.

Here is the call graph for this function:


hash_nt_password_hash
generate_authenticator_response_pwhash
sha1_vector
md4_vector
generate_authenticator_response
nt_password_hash

6.105.2.3 void generate_authenticator_response_pwhash (const u8 ∗ *password_hash*, const u8 ∗ *peer_challenge*, const u8 ∗ *auth_challenge*, const u8 ∗ *username*, size_t *username_len*, const u8 ∗ *nt_response*, u8 ∗ *response*)

GenerateAuthenticatorResponse() - RFC 2759, Sect. 8.7.

Parameters:

password_hash 16-octet PasswordHash (IN)

nt_response 24-octet NT-Response (IN)

peer_challenge 16-octet PeerChallenge (IN)

auth_challenge 16-octet AuthenticatorChallenge (IN)

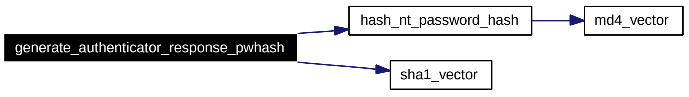
username 0-to-256-char UserName (IN)

username_len Length of username

response 20-octet AuthenticatorResponse (OUT) (note: this value is usually encoded as a 42-octet ASCII string (S=<hexdump of="" response="">)

Definition at line 177 of file ms_funcs.c.

Here is the call graph for this function:



6.105.2.4 void generate_nt_response (const u8 ∗ *auth_challenge*, const u8 ∗ *peer_challenge*, const u8 ∗ *username*, size_t *username_len*, const u8 ∗ *password*, size_t *password_len*, u8 ∗ *response*)

GenerateNTResponse() - RFC 2759, Sect. 8.1.

Parameters:

auth_challenge 16-octet AuthenticatorChallenge (IN)

peer_hallenge 16-octet PeerChallenge (IN)

username 0-to-256-char UserName (IN)

username_len Length of username

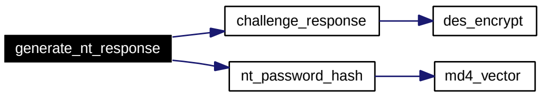
password 0-to-256-unicode-char Password (IN; ASCII)

password_len Length of password

response 24-octet Response (OUT)

Definition at line 126 of file ms_funcs.c.

Here is the call graph for this function:



6.105.2.5 void generate_nt_response_pwhash (const u8 ∗ *auth_challenge*, const u8 ∗ *peer_challenge*, const u8 ∗ *username*, size_t *username_len*, const u8 ∗ *password_hash*, u8 ∗ *response*)

GenerateNTResponse() - RFC 2759, Sect. 8.1.

Parameters:

auth_challenge 16-octet AuthenticatorChallenge (IN)

peer_hallenge 16-octet PeerChallenge (IN)

username 0-to-256-char UserName (IN)

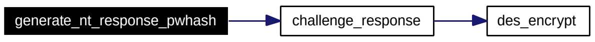
username_len Length of username

password_hash 16-octet PasswordHash (IN)

response 24-octet Response (OUT)

Definition at line 151 of file ms_funcs.c.

Here is the call graph for this function:



6.105.2.6 void get_asymetric_start_key (const u8 ∗ *master_key*, u8 ∗ *session_key*, size_t *session_key_len*, int *is_send*, int *is_server*)

GetAsymetricStartKey() - RFC 3079, Sect. 3.4.

Parameters:

master_key 16-octet MasterKey (IN)

session_key 8-to-16 octet SessionKey (OUT)

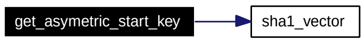
session_key_len SessionKeyLength (Length of session_key) (IN)

is_send IsSend (IN, BOOLEAN)

is_server IsServer (IN, BOOLEAN)

Definition at line 302 of file ms_funcs.c.

Here is the call graph for this function:



6.105.2.7 void get_master_key (const u8 ∗ *password_hash_hash*, const u8 ∗ *nt_response*, u8 ∗ *master_key*)

GetMasterKey() - RFC 3079, Sect. 3.4.

Parameters:

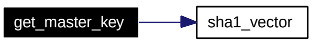
password_hash_hash 16-octet PasswordHashHash (IN)

nt_response 24-octet NTResponse (IN)

master_key 16-octet MasterKey (OUT)

Definition at line 272 of file ms_funcs.c.

Here is the call graph for this function:



6.105.2.8 void hash_nt_password_hash (const u8 ∗ *password_hash*, u8 ∗ *password_hash_hash*)

HashNtPasswordHash() - RFC 2759, Sect. 8.4.

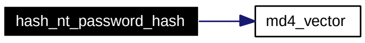
Parameters:

password_hash 16-octet PasswordHash (IN)

password_hash_hash 16-octet PasswordHashHash (OUT)

Definition at line 88 of file ms_funcs.c.

Here is the call graph for this function:



6.105.2.9 void new_password_encrypted_with_old_nt_password_hash (const u8 ∗ *new_password*, size_t *new_password_len*, const u8 ∗ *old_password*, size_t *old_password_len*, u8 ∗ *encrypted_pw_block*)

NewPasswordEncryptedWithOldNtPasswordHash() - RFC 2759, Sect. 8.9.

Parameters:

new_password 0-to-256-unicode-char NewPassword (IN; ASCII)

new_password_len Length of new_password

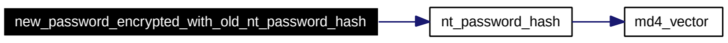
old_password 0-to-256-unicode-char OldPassword (IN; ASCII)

old_password_len Length of old_password

encrypted_pw_block 516-octet EncryptedPwBlock (OUT)

Definition at line 406 of file ms_funcs.c.

Here is the call graph for this function:



6.105.2.10 void nt_challenge_response (const u8 ∗ *challenge*, const u8 ∗ *password*, size_t *password_len*, u8 ∗ *response*)

NtChallengeResponse() - RFC 2433, Sect. A.5.

Parameters:

challenge 8-octet Challenge (IN)

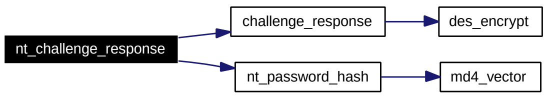
password 0-to-256-unicode-char Password (IN; ASCII)

password_len Length of password

response 24-octet Response (OUT)

Definition at line 256 of file ms_funcs.c.

Here is the call graph for this function:



6.105.2.11 void nt_password_hash (const u8 ∗ *password*, size_t *password_len*, u8 ∗ *password_hash*)

NtPasswordHash() - RFC 2759, Sect. 8.3.

Parameters:

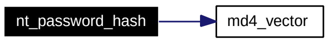
password 0-to-256-unicode-char Password (IN; ASCII)

password_len Length of password

password_hash 16-octet PasswordHash (OUT)

Definition at line 61 of file ms_funcs.c.

Here is the call graph for this function:



6.105.2.12 void old_nt_password_hash_encrypted_with_new_nt_password_hash (const u8 ∗ *new_password*, size_t *new_password_len*, const u8 ∗ *old_password*, size_t *old_password_len*, u8 ∗ *encrypted_password_hash*)

OldNtPasswordHashEncryptedWithNewNtPasswordHash() - RFC 2759, Sect. 8.12.

Parameters:

new_password 0-to-256-unicode-char NewPassword (IN; ASCII)

new_password_len Length of new_password

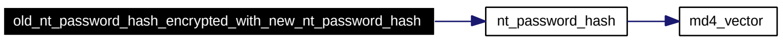
old_password 0-to-256-unicode-char OldPassword (IN; ASCII)

old_password_len Length of old_password

encrypted_password_ash 16-octet EncryptedPasswordHash (OUT)

Definition at line 444 of file ms_funcs.c.

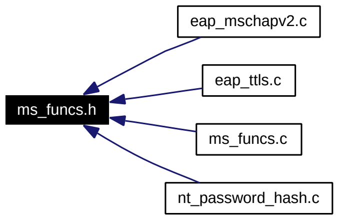
Here is the call graph for this function:



6.106 ms_funcs.h File Reference

WPA Supplicant / shared MSCHAPV2 helper functions / RFC 2433 / RFC 2759.

This graph shows which files directly or indirectly include this file:



Functions

- void generate_nt_response (const u8 ∗auth_challenge, const u8 ∗peer_challenge, const u8 ∗username, size_t username_len, const u8 ∗password, size_t password_len, u8 ∗response)

  GenerateNTResponse() - RFC 2759, Sect. 8.1.

- void generate_nt_response_pwhash (const u8 ∗auth_challenge, const u8 ∗peer_challenge, const u8 ∗username, size_t username_len, const u8 ∗password_hash, u8 ∗response)

  GenerateNTResponse() - RFC 2759, Sect. 8.1.

- void generate_authenticator_response (const u8 ∗password, size_t password_len, const u8 ∗peer_-challenge, const u8 ∗auth_challenge, const u8 ∗username, size_t username_len, const u8 ∗nt_-response, u8 ∗response)

  GenerateAuthenticatorResponse() - RFC 2759, Sect. 8.7.

- void generate_authenticator_response_pwhash (const u8 ∗password_hash, const u8 ∗peer_-challenge, const u8 ∗auth_challenge, const u8 ∗username, size_t username_len, const u8 ∗nt_-response, u8 ∗response)

  GenerateAuthenticatorResponse() - RFC 2759, Sect. 8.7.

- void nt_challenge_response (const u8 ∗challenge, const u8 ∗password, size_t password_len, u8 ∗response)

  NtChallengeResponse() - RFC 2433, Sect. A.5.

- void challenge_response (const u8 ∗challenge, const u8 ∗password_hash, u8 ∗response)

  ChallengeResponse() - RFC 2759, Sect. 8.5.

- void nt_password_hash (const u8 ∗password, size_t password_len, u8 ∗password_hash)

  NtPasswordHash() - RFC 2759, Sect. 8.3.

- void hash_nt_password_hash (const u8 ∗password_hash, u8 ∗password_hash_hash)

  HashNtPasswordHash() - RFC 2759, Sect. 8.4.

- void get_master_key (const u8 ∗password_hash_hash, const u8 ∗nt_response, u8 ∗master_key)

  GetMasterKey() - RFC 3079, Sect. 3.4.

- void get_asymetric_start_key (const u8 ∗master_key, u8 ∗session_key, size_t session_key_len, int is_send, int is_server)

    GetAsymetricStartKey() - RFC 3079, Sect. 3.4.

- void new_password_encrypted_with_old_nt_password_hash (const u8 ∗new_password, size_t new_password_len, const u8 ∗old_password, size_t old_password_len, u8 ∗encrypted_pw_block)

    NewPasswordEncryptedWithOldNtPasswordHash() - RFC 2759, Sect. 8.9.

- void old_nt_password_hash_encrypted_with_new_nt_password_hash (const u8 ∗new_password, size_t new_password_len, const u8 ∗old_password, size_t old_password_len, u8 ∗encrypted_password_hash)

    OldNtPasswordHashEncryptedWithNewNtPasswordHash() - RFC 2759, Sect. 8.12.

6.106.1 Detailed Description

WPA Supplicant / shared MSCHAPV2 helper functions / RFC 2433 / RFC 2759.

Copyright

Definition in file ms_funcs.h.

6.106.2 Function Documentation

6.106.2.1 void challenge_response (const u8 ∗ *challenge*, const u8 ∗ *password_hash*, u8 ∗ *response*)

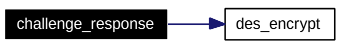
ChallengeResponse() - RFC 2759, Sect. 8.5.

Parameters:

challenge	8-octet Challenge (IN)
password_hash	16-octet PasswordHash (IN)
response	24-octet Response (OUT)

Definition at line 102 of file ms_funcs.c.

Here is the call graph for this function:



6.106.2.2 void generate_authenticator_response (const u8 ∗ *password*, size_t *password_len*, const u8 ∗ *peer_challenge*, const u8 ∗ *auth_challenge*, const u8 ∗ *username*, size_t *username_len*, const u8 ∗ *nt_response*, u8 ∗ *response*)

GenerateAuthenticatorResponse() - RFC 2759, Sect. 8.7.

Parameters:

password 0-to-256-unicode-char Password (IN; ASCII)

password_len Length of password

nt_response 24-octet NT-Response (IN)

peer_challenge 16-octet PeerChallenge (IN)

auth_challenge 16-octet AuthenticatorChallenge (IN)

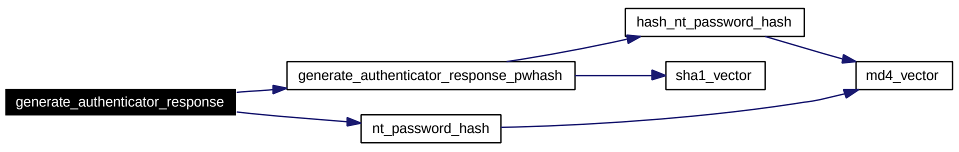
username 0-to-256-char UserName (IN)

username_len Length of username

response 20-octet AuthenticatorResponse (OUT) (note: this value is usually encoded as a 42-octet ASCII string (S=<hexdump of="" response="">)

Definition at line 233 of file ms_funcs.c.

Here is the call graph for this function:


generate_authenticator_response
generate_authenticator_response_pwhash
hash_nt_password_hash
sha1_vector
md4_vector
nt_password_hash

6.106.2.3 void generate_authenticator_response_pwhash (const u8 ∗ *password_hash*, const u8 ∗ *peer_challenge*, const u8 ∗ *auth_challenge*, const u8 ∗ *username*, size_t *username_len*, const u8 ∗ *nt_response*, u8 ∗ *response*)

GenerateAuthenticatorResponse() - RFC 2759, Sect. 8.7.

Parameters:

password_hash 16-octet PasswordHash (IN)

nt_response 24-octet NT-Response (IN)

peer_challenge 16-octet PeerChallenge (IN)

auth_challenge 16-octet AuthenticatorChallenge (IN)

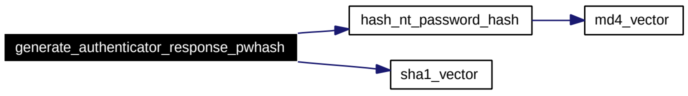
username 0-to-256-char UserName (IN)

username_len Length of username

response 20-octet AuthenticatorResponse (OUT) (note: this value is usually encoded as a 42-octet ASCII string (S=<hexdump of="" response="">)

Definition at line 177 of file ms_funcs.c.

Here is the call graph for this function:



6.106.2.4 void generate_nt_response (const u8 ∗ *auth_challenge*, const u8 ∗ *peer_challenge*, const u8 ∗ *username*, size_t *username_len*, const u8 ∗ *password*, size_t *password_len*, u8 ∗ *response*)

GenerateNTResponse() - RFC 2759, Sect. 8.1.

Parameters:

auth_challenge 16-octet AuthenticatorChallenge (IN)

peer_hallenge 16-octet PeerChallenge (IN)

username 0-to-256-char UserName (IN)

username_len Length of username

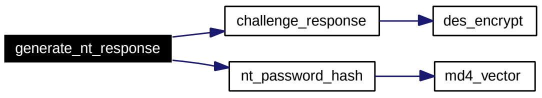
password 0-to-256-unicode-char Password (IN; ASCII)

password_len Length of password

response 24-octet Response (OUT)

Definition at line 126 of file ms_funcs.c.

Here is the call graph for this function:



6.106.2.5 void generate_nt_response_pwhash (const u8 ∗ *auth_challenge*, const u8 ∗ *peer_challenge*, const u8 ∗ *username*, size_t *username_len*, const u8 ∗ *password_hash*, u8 ∗ *response*)

GenerateNTResponse() - RFC 2759, Sect. 8.1.

Parameters:

auth_challenge 16-octet AuthenticatorChallenge (IN)

peer_hallenge 16-octet PeerChallenge (IN)

username 0-to-256-char UserName (IN)

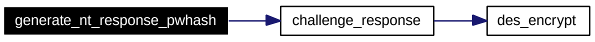
username_len Length of username

password_hash 16-octet PasswordHash (IN)

response 24-octet Response (OUT)

Definition at line 151 of file ms_funcs.c.

Here is the call graph for this function:



6.106.2.6 void get_asymetric_start_key (const u8 ∗ *master_key*, u8 ∗ *session_key*, size_t *session_key_len*, int *is_send*, int *is_server*)

GetAsymetricStartKey() - RFC 3079, Sect. 3.4.

Parameters:

master_key 16-octet MasterKey (IN)

session_key 8-to-16 octet SessionKey (OUT)

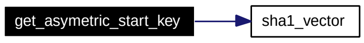
session_key_len SessionKeyLength (Length of session_key) (IN)

is_send IsSend (IN, BOOLEAN)

is_server IsServer (IN, BOOLEAN)

Definition at line 302 of file ms_funcs.c.

Here is the call graph for this function:



6.106.2.7 void get_master_key (const u8 ∗ *password_hash_hash*, const u8 ∗ *nt_response*, u8 ∗ *master_key*)

GetMasterKey() - RFC 3079, Sect. 3.4.

Parameters:

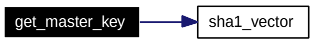
password_hash_hash 16-octet PasswordHashHash (IN)

nt_response 24-octet NTResponse (IN)

master_key 16-octet MasterKey (OUT)

Definition at line 272 of file ms_funcs.c.

Here is the call graph for this function:



6.106.2.8 void hash_nt_password_hash (const u8 ∗ *password_hash*, u8 ∗ *password_hash_hash*)

HashNtPasswordHash() - RFC 2759, Sect. 8.4.

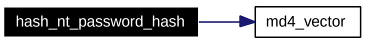
Parameters:

password_hash 16-octet PasswordHash (IN)

password_hash_hash 16-octet PasswordHashHash (OUT)

Definition at line 88 of file ms_funcs.c.

Here is the call graph for this function:



6.106.2.9 void new_password_encrypted_with_old_nt_password_hash (const u8 ∗ *new_password*, size_t *new_password_len*, const u8 ∗ *old_password*, size_t *old_password_len*, u8 ∗ *encrypted_pw_block*)

NewPasswordEncryptedWithOldNtPasswordHash() - RFC 2759, Sect. 8.9.

Parameters:

new_password 0-to-256-unicode-char NewPassword (IN; ASCII)

new_password_len Length of new_password

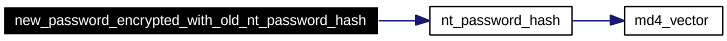
old_password 0-to-256-unicode-char OldPassword (IN; ASCII)

old_password_len Length of old_password

encrypted_pw_block 516-octet EncryptedPwBlock (OUT)

Definition at line 406 of file ms_funcs.c.

Here is the call graph for this function:



6.106.2.10 void nt_challenge_response (const u8 ∗ *challenge*, const u8 ∗ *password*, size_t *password_len*, u8 ∗ *response*)

NtChallengeResponse() - RFC 2433, Sect. A.5.

Parameters:

challenge 8-octet Challenge (IN)

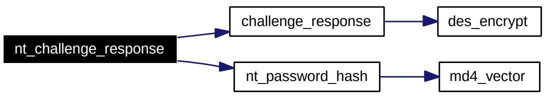
password 0-to-256-unicode-char Password (IN; ASCII)

password_len Length of password

response 24-octet Response (OUT)

Definition at line 256 of file ms_funcs.c.

Here is the call graph for this function:



6.106.2.11 void nt_password_hash (const u8 ∗ *password*, size_t *password_len*, u8 ∗ *password_hash*)

NtPasswordHash() - RFC 2759, Sect. 8.3.

Parameters:

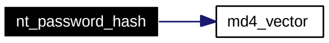
password 0-to-256-unicode-char Password (IN; ASCII)

password_len Length of password

password_hash 16-octet PasswordHash (OUT)

Definition at line 61 of file ms_funcs.c.

Here is the call graph for this function:



6.106.2.12 void old_nt_password_hash_encrypted_with_new_nt_password_hash (const u8 ∗ *new_password*, size_t *new_password_len*, const u8 ∗ *old_password*, size_t *old_password_len*, u8 ∗ *encrypted_password_hash*)

OldNtPasswordHashEncryptedWithNewNtPasswordHash() - RFC 2759, Sect. 8.12.

Parameters:

new_password 0-to-256-unicode-char NewPassword (IN; ASCII)

new_password_len Length of new_password

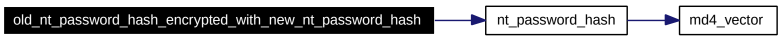
old_password 0-to-256-unicode-char OldPassword (IN; ASCII)

old_password_len Length of old_password

encrypted_password_ash 16-octet EncryptedPasswordHash (OUT)

Definition at line 444 of file ms_funcs.c.

Here is the call graph for this function:

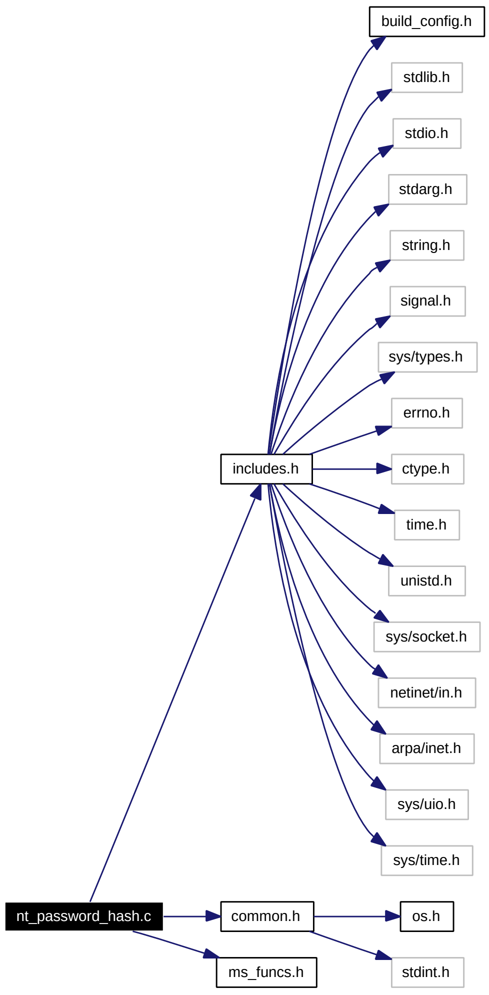


6.107 nt_password_hash.c File Reference

hostapd - Plaintext password to NtPasswordHash

```
#include "includes.h"
#include "common.h"
#include "ms_funcs.h"
```

Include dependency graph for nt_password_hash.c:



Functions

- int **main** (int argc, char ∗argv[ ])

6.107.1 Detailed Description

hostapd - Plaintext password to NtPasswordHash

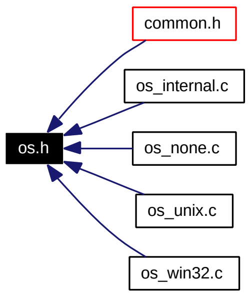
Copyright

Definition in file nt_password_hash.c.

6.108 os.h File Reference

wpa_supplicant/hostapd / OS specific functions

This graph shows which files directly or indirectly include this file:



Defines

- #define **os_time_before**(a, b)
- #define **os_time_sub**(a, b, res)
- #define **os_malloc**(s) malloc((s))
- #define **os_realloc**(p, s) realloc((p), (s))
- #define **os_free**(p) free((p))
- #define **os_memcpy**(d, s, n) memcpy((d), (s), (n))
- #define **os_memmove**(d, s, n) memmove((d), (s), (n))
- #define **os_memset**(s, c, n) memset(s, c, n)
- #define **os_memcmp**(s1, s2, n) memcmp((s1), (s2), (n))
- #define **os_strdup**(s) strdup(s)
- #define **os_strlen**(s) strlen(s)
- #define **os_strcasecmp**(s1, s2) strcasecmp((s1), (s2))
- #define **os_strncasecmp**(s1, s2, n) strncasecmp((s1), (s2), (n))
- #define **os_strchr**(s, c) strchr((s), (c))
- #define **os_strcmp**(s1, s2) strcmp((s1), (s2))
- #define **os_strncmp**(s1, s2, n) strncmp((s1), (s2), (n))
- #define **os_strncpy**(d, s, n) strncpy((d), (s), (n))
- #define **os_strrchr**(s, c) strrchr((s), (c))
- #define **os_strstr**(h, n) strstr((h), (n))
- #define **os_snprintf** snprintf

Typedefs

- typedef long **os_time_t**

Functions

- void os_sleep (os_time_t sec, os_time_t usec)

  Sleep (sec, usec).

- int os_get_time (struct os_time ∗t)

Get current time (sec, usec).

- int os_mktime (int year, int month, int day, int hour, int min, int sec, os_time_t ∗t)
  Convert broken-down time into seconds since 1970-01-01.
- int os_daemonize (const char ∗pid_file)
  Run in the background (detach from the controlling terminal).
- void os_daemonize_terminate (const char ∗pid_file)
  Stop running in the background (remove pid file).
- int os_get_random (unsigned char ∗buf, size_t len)
  Get cryptographically strong pseudo random data.
- unsigned long os_random (void)
  Get pseudo random value (not necessarily very strong).
- char ∗ os_rel2abs_path (const char ∗rel_path)
  Get an absolute path for a file.
- int os_program_init (void)
  Program initialization (called at start).
- void os_program_deinit (void)
  Program deinitialization (called just before exit).
- int os_setenv (const char ∗name, const char ∗value, int overwrite)
  Set environment variable.
- int os_unsetenv (const char ∗name)
  Delete environent variable.
- char ∗ os_readfile (const char ∗name, size_t ∗len)
  Read a file to an allocated memory buffer.
- void ∗ os_zalloc (size_t size)
  Allocate and zero memory.

6.108.1 Detailed Description

wpa_supplicant/hostapd / OS specific functions

Copyright

Copyright (c) 2005-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file os.h.

6.108.2 Define Documentation

6.108.2.1 #define os_time_before(a, b)

Value:

```
((a)->sec < (b)->sec || \
         ((a)->sec == (b)->sec && (a)->usec < (b)->usec))
```

Definition at line 45 of file os.h.

6.108.2.2 #define os_time_sub(a, b, res)

Value:

```
do { \
        (res)->sec = (a)->sec - (b)->sec; \
        (res)->usec = (a)->usec - (b)->usec; \
        if ((res)->usec < 0) { \
                (res)->sec--; \
                (res)->usec += 1000000; \
        } \
} while (0)
```

Definition at line 49 of file os.h.

6.108.3 Function Documentation

6.108.3.1 int os_daemonize (const char ∗ *pid_file*)

Run in the background (detach from the controlling terminal).

Parameters:
 pid_file File name to write the process ID to or NULL to skip this

Returns:
 0 on success, -1 on failure

Definition at line 74 of file os_internal.c.

6.108.3.2 void os_daemonize_terminate (const char ∗ *pid_file*)

Stop running in the background (remove pid file).

Parameters:
 pid_file File name to write the process ID to or NULL to skip this

Definition at line 93 of file os_internal.c.

6.108.3.3 int os_get_random (unsigned char $*$ *buf*, size_t *len*)

Get cryptographically strong pseudo random data.

Parameters:

buf Buffer for pseudo random data

len Length of the buffer

Returns:

0 on success, -1 on failure

Definition at line 100 of file os_internal.c.

6.108.3.4 int os_get_time (struct os_time $*$ *t*)

Get current time (sec, usec).

Parameters:

t Pointer to buffer for the time

Returns:

0 on success, -1 on failure

Definition at line 40 of file os_internal.c.

6.108.3.5 int os_mktime (int *year*, int *month*, int *day*, int *hour*, int *min*, int *sec*, os_time_t $*$ *t*)

Convert broken-down time into seconds since 1970-01-01.

Parameters:

year Four digit year

month Month (1 .. 12)

day Day of month (1 .. 31)

hour Hour (0 .. 23)

min Minute (0 .. 59)

sec Second (0 .. 60)

t Buffer for returning calendar time representation (seconds since 1970-01-01 00:00:00)

Returns:

0 on success, -1 on failure

Definition at line 51 of file os_internal.c.

6.108.3.6 void os_program_deinit (void)

Program deinitialization (called just before exit).

This function is called just before a program exists. If there are any OS specific processing, e.g., freeing resourced allocated in os_program_init(), it should be done here. It is also acceptable for this function to do nothing.

Definition at line 169 of file os_internal.c.

6.108.3.7 int os_program_init (void)

Program initialization (called at start).

Returns:
0 on success, -1 on failure

This function is called when a programs starts. If there are any OS specific processing that is needed, it can be placed here. It is also acceptable to just return 0 if not special processing is needed.

Definition at line 163 of file os_internal.c.

6.108.3.8 unsigned long os_random (void)

Get pseudo random value (not necessarily very strong).

Returns:
Pseudo random value

Definition at line 118 of file os_internal.c.

6.108.3.9 char∗ os_readfile (const char ∗ *name*, size_t ∗ *len*)

Read a file to an allocated memory buffer.

Parameters:
name Name of the file to read
len For returning the length of the allocated buffer

Returns:
Pointer to the allocated buffer or NULL on failure

This function allocates memory and reads the given file to this buffer. Both binary and text files can be read with this function. The caller is responsible for freeing the returned buffer with os_free().

Definition at line 191 of file os_internal.c.

6.108.3.10 char∗ os_rel2abs_path (const char ∗ *rel_path*)

Get an absolute path for a file.

Parameters:
rel_path Relative path to a file

Returns:
Absolute path for the file or NULL on failure

This function tries to convert a relative path of a file to an absolute path in order for the file to be found even if current working directory has changed. The returned value is allocated and caller is responsible for freeing it. It is acceptable to just return the same path in an allocated buffer, e.g., return strdup(rel_path). This function is only used to find configuration files when os_daemonize() may have changed the current working directory and relative path would be pointing to a different location.

Definition at line 124 of file os_internal.c.

6.108.3.11 int os_setenv (const char ∗ *name*, const char ∗ *value*, int *overwrite*)

Set environment variable.

Parameters:
- ***name*** Name of the variable
- ***value*** Value to set to the variable
- ***overwrite*** Whether existing variable should be overwritten

Returns:
0 on success, -1 on error

This function is only used for wpa_cli action scripts. OS wrapper does not need to implement this if such functionality is not needed.

Definition at line 174 of file os_internal.c.

6.108.3.12 void os_sleep (os_time_t *sec*, os_time_t *usec*)

Sleep (sec, usec).

Parameters:
- ***sec*** Number of seconds to sleep
- ***usec*** Number of microseconds to sleep

Definition at line 31 of file os_internal.c.

6.108.3.13 int os_unsetenv (const char ∗ *name*)

Delete environent variable.

Parameters:
- ***name*** Name of the variable

Returns:
0 on success, -1 on error

This function is only used for wpa_cli action scripts. OS wrapper does not need to implement this if such functionality is not needed.

Definition at line 180 of file os_internal.c.

6.108.3.14 void∗ os_zalloc (size_t *size*)

Allocate and zero memory.

Parameters:
- ***size*** Number of bytes to allocate

Returns:
Pointer to allocated and zeroed memory or NULL on failure

Caller is responsible for freeing the returned buffer with os_free().

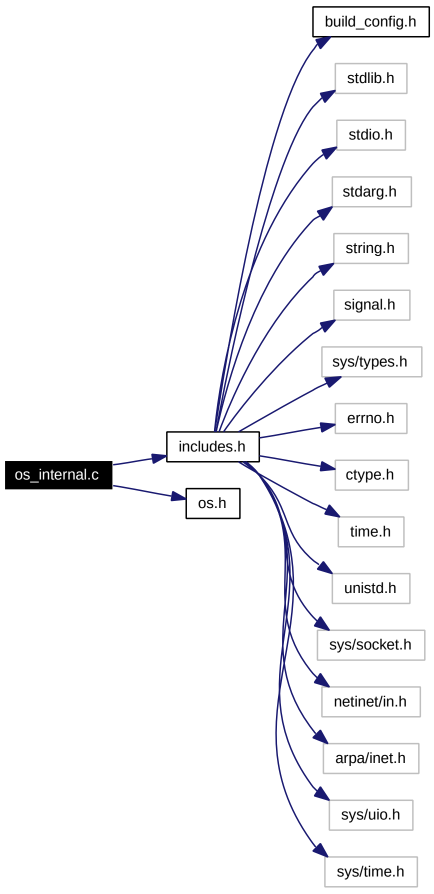
Definition at line 217 of file os_internal.c.

6.109 os_internal.c File Reference

wpa_supplicant/hostapd / Internal implementation of OS specific functions

```
#include "includes.h"
#include "os.h"
```

Include dependency graph for os_internal.c:



Functions

- void os_sleep (os_time_t sec, os_time_t usec)

    Sleep (sec, usec).

- int os_get_time (struct os_time ∗t)

    Get current time (sec, usec).

- int os_mktime (int year, int month, int day, int hour, int min, int sec, os_time_t ∗t)

    Convert broken-down time into seconds since 1970-01-01.

- int os_daemonize (const char ∗pid_file)

    Run in the background (detach from the controlling terminal).

- void os_daemonize_terminate (const char ∗pid_file)

  Stop running in the background (remove pid file).

- int os_get_random (unsigned char ∗buf, size_t len)

  Get cryptographically strong pseudo random data.

- unsigned long os_random (void)

  Get pseudo random value (not necessarily very strong).

- char ∗ os_rel2abs_path (const char ∗rel_path)

  Get an absolute path for a file.

- int os_program_init (void)

  Program initialization (called at start).

- void os_program_deinit (void)

  Program deinitialization (called just before exit).

- int os_setenv (const char ∗name, const char ∗value, int overwrite)

  Set environment variable.

- int os_unsetenv (const char ∗name)

  Delete environent variable.

- char ∗ os_readfile (const char ∗name, size_t ∗len)

  Read a file to an allocated memory buffer.

- void ∗ os_zalloc (size_t size)

  Allocate and zero memory.

- void ∗ **os_malloc** (size_t size)
- void ∗ **os_realloc** (void ∗ptr, size_t size)
- void **os_free** (void ∗ptr)
- void ∗ **os_memcpy** (void ∗dest, const void ∗src, size_t n)
- void ∗ **os_memmove** (void ∗dest, const void ∗src, size_t n)
- void ∗ **os_memset** (void ∗s, int c, size_t n)
- int **os_memcmp** (const void ∗s1, const void ∗s2, size_t n)
- char ∗ **os_strdup** (const char ∗s)
- size_t **os_strlen** (const char ∗s)
- int **os_strcasecmp** (const char ∗s1, const char ∗s2)
- int **os_strncasecmp** (const char ∗s1, const char ∗s2, size_t n)
- char ∗ **os_strchr** (const char ∗s, int c)
- char ∗ **os_strrchr** (const char ∗s, int c)
- int **os_strcmp** (const char ∗s1, const char ∗s2)
- int **os_strncmp** (const char ∗s1, const char ∗s2, size_t n)
- char ∗ **os_strncpy** (char ∗dest, const char ∗src, size_t n)
- char ∗ **os_strstr** (const char ∗haystack, const char ∗needle)
- int **os_snprintf** (char ∗str, size_t size, const char ∗format,...)

6.109.1 Detailed Description

wpa_supplicant/hostapd / Internal implementation of OS specific functions

Copyright

Copyright (c) 2005-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This file is an example of operating system specific wrapper functions. This version implements many of the functions internally, so it can be used to fill in missing functions from the target system C libraries.

Some of the functions are using standard C library calls in order to keep this file in working condition to allow the functions to be tested on a Linux target. Please note that OS_NO_C_LIB_DEFINES needs to be defined for this file to work correctly. Note that these implementations are only examples and are not optimized for speed.

Definition in file os_internal.c.

6.109.2 Function Documentation

6.109.2.1 int os_daemonize (const char ∗ *pid_file*)

Run in the background (detach from the controlling terminal).

Parameters:

pid_file File name to write the process ID to or NULL to skip this

Returns:

0 on success, -1 on failure

Definition at line 74 of file os_internal.c.

6.109.2.2 void os_daemonize_terminate (const char ∗ *pid_file*)

Stop running in the background (remove pid file).

Parameters:

pid_file File name to write the process ID to or NULL to skip this

Definition at line 93 of file os_internal.c.

6.109.2.3 int os_get_random (unsigned char ∗ *buf*, size_t *len*)

Get cryptographically strong pseudo random data.

Parameters:

buf Buffer for pseudo random data

len Length of the buffer

Returns:
0 on success, -1 on failure

Definition at line 100 of file os_internal.c.

6.109.2.4 int os_get_time (struct os_time ∗ *t*)

Get current time (sec, usec).

Parameters:
t Pointer to buffer for the time

Returns:
0 on success, -1 on failure

Definition at line 40 of file os_internal.c.

6.109.2.5 int os_mktime (int *year*, int *month*, int *day*, int *hour*, int *min*, int *sec*, os_time_t ∗ *t*)

Convert broken-down time into seconds since 1970-01-01.

Parameters:
year Four digit year

month Month (1 .. 12)

day Day of month (1 .. 31)

hour Hour (0 .. 23)

min Minute (0 .. 59)

sec Second (0 .. 60)

t Buffer for returning calendar time representation (seconds since 1970-01-01 00:00:00)

Returns:
0 on success, -1 on failure

Definition at line 51 of file os_internal.c.

6.109.2.6 void os_program_deinit (void)

Program deinitialization (called just before exit).

This function is called just before a program exists. If there are any OS specific processing, e.g., freeing resourced allocated in os_program_init(), it should be done here. It is also acceptable for this function to do nothing.

Definition at line 169 of file os_internal.c.

6.109.2.7 int os_program_init (void)

Program initialization (called at start).

Returns:

0 on success, -1 on failure

This function is called when a programs starts. If there are any OS specific processing that is needed, it can be placed here. It is also acceptable to just return 0 if not special processing is needed.

Definition at line 163 of file os_internal.c.

6.109.2.8 unsigned long os_random (void)

Get pseudo random value (not necessarily very strong).

Returns:

Pseudo random value

Definition at line 118 of file os_internal.c.

6.109.2.9 char∗ os_readfile (const char ∗ *name*, size_t ∗ *len*)

Read a file to an allocated memory buffer.

Parameters:

name Name of the file to read

len For returning the length of the allocated buffer

Returns:

Pointer to the allocated buffer or NULL on failure

This function allocates memory and reads the given file to this buffer. Both binary and text files can be read with this function. The caller is responsible for freeing the returned buffer with os_free().

Definition at line 191 of file os_internal.c.

6.109.2.10 char∗ os_rel2abs_path (const char ∗ *rel_path*)

Get an absolute path for a file.

Parameters:

rel_path Relative path to a file

Returns:

Absolute path for the file or NULL on failure

This function tries to convert a relative path of a file to an absolute path in order for the file to be found even if current working directory has changed. The returned value is allocated and caller is responsible for freeing it. It is acceptable to just return the same path in an allocated buffer, e.g., return strdup(rel_path). This function is only used to find configuration files when os_daemonize() may have changed the current working directory and relative path would be pointing to a different location.

Definition at line 124 of file os_internal.c.

6.109.2.11 int os_setenv (const char ∗ *name*, const char ∗ *value*, int *overwrite*)

Set environment variable.

Parameters:

name Name of the variable

value Value to set to the variable

overwrite Whether existing variable should be overwritten

Returns:

0 on success, -1 on error

This function is only used for wpa_cli action scripts. OS wrapper does not need to implement this if such functionality is not needed.

Definition at line 174 of file os_internal.c.

6.109.2.12 void os_sleep (os_time_t *sec*, os_time_t *usec*)

Sleep (sec, usec).

Parameters:

sec Number of seconds to sleep

usec Number of microseconds to sleep

Definition at line 31 of file os_internal.c.

6.109.2.13 int os_unsetenv (const char ∗ *name*)

Delete environent variable.

Parameters:

name Name of the variable

Returns:

0 on success, -1 on error

This function is only used for wpa_cli action scripts. OS wrapper does not need to implement this if such functionality is not needed.

Definition at line 180 of file os_internal.c.

6.109.2.14 void∗ os_zalloc (size_t *size*)

Allocate and zero memory.

Parameters:

size Number of bytes to allocate

Returns:

Pointer to allocated and zeroed memory or NULL on failure

Caller is responsible for freeing the returned buffer with os_free().

Definition at line 217 of file os_internal.c.

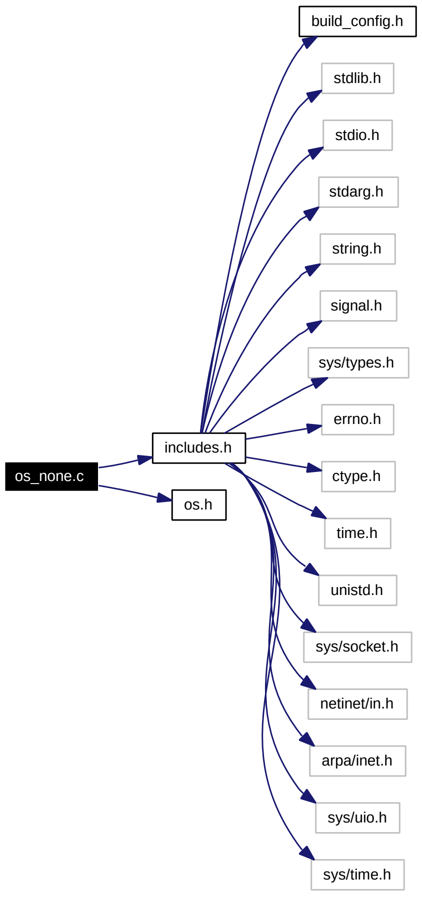
6.110 os_none.c File Reference

wpa_supplicant/hostapd / Empty OS specific functions

`#include "includes.h"`

`#include "os.h"`

Include dependency graph for os_none.c:



Functions

- void os_sleep (os_time_t sec, os_time_t usec)

  Sleep (sec, usec).

- int os_get_time (struct os_time ∗t)

  Get current time (sec, usec).

- int os_mktime (int year, int month, int day, int hour, int min, int sec, os_time_t ∗t)

  Convert broken-down time into seconds since 1970-01-01.

- int os_daemonize (const char ∗pid_file)

  Run in the background (detach from the controlling terminal).

- void os_daemonize_terminate (const char ∗pid_file)

  Stop running in the background (remove pid file).

- int os_get_random (unsigned char ∗buf, size_t len)

  Get cryptographically strong pseudo random data.

- unsigned long os_random (void)

  Get pseudo random value (not necessarily very strong).

- char ∗ os_rel2abs_path (const char ∗rel_path)

  Get an absolute path for a file.

- int os_program_init (void)

  Program initialization (called at start).

- void os_program_deinit (void)

  Program deinitialization (called just before exit).

- int os_setenv (const char ∗name, const char ∗value, int overwrite)

  Set environment variable.

- int os_unsetenv (const char ∗name)

  Delete environent variable.

- char ∗ os_readfile (const char ∗name, size_t ∗len)

  Read a file to an allocated memory buffer.

- void ∗ os_zalloc (size_t size)

  Allocate and zero memory.

6.110.1 Detailed Description

wpa_supplicant/hostapd / Empty OS specific functions

Copyright

This file can be used as a starting point when adding a new OS target. The functions here do not really work as-is since they are just empty or only return an error value. os_internal.c can be used as another starting point or reference since it has example implementation of many of these functions.

Definition in file os_none.c.

6.110.2 Function Documentation

6.110.2.1 int os_daemonize (const char ∗ *pid_file*)

Run in the background (detach from the controlling terminal).

Parameters:

pid_file File name to write the process ID to or NULL to skip this

Returns:

0 on success, -1 on failure

Definition at line 43 of file os_none.c.

6.110.2.2 void os_daemonize_terminate (const char ∗ *pid_file*)

Stop running in the background (remove pid file).

Parameters:

pid_file File name to write the process ID to or NULL to skip this

Definition at line 49 of file os_none.c.

6.110.2.3 int os_get_random (unsigned char ∗ *buf*, size_t *len*)

Get cryptographically strong pseudo random data.

Parameters:

buf Buffer for pseudo random data

len Length of the buffer

Returns:

0 on success, -1 on failure

Definition at line 54 of file os_none.c.

6.110.2.4 int os_get_time (struct os_time ∗ *t*)

Get current time (sec, usec).

Parameters:

t Pointer to buffer for the time

Returns:

0 on success, -1 on failure

Definition at line 30 of file os_none.c.

6.110.2.5 int os_mktime (int *year*, int *month*, int *day*, int *hour*, int *min*, int *sec*, os_time_t * *t*)

Convert broken-down time into seconds since 1970-01-01.

Parameters:

year Four digit year

month Month (1 .. 12)

day Day of month (1 .. 31)

hour Hour (0 .. 23)

min Minute (0 .. 59)

sec Second (0 .. 60)

t Buffer for returning calendar time representation (seconds since 1970-01-01 00:00:00)

Returns:

0 on success, -1 on failure

Definition at line 36 of file os_none.c.

6.110.2.6 void os_program_deinit (void)

Program deinitialization (called just before exit).

This function is called just before a program exists. If there are any OS specific processing, e.g., freeing resourced allocated in os_program_init(), it should be done here. It is also acceptable for this function to do nothing.

Definition at line 78 of file os_none.c.

6.110.2.7 int os_program_init (void)

Program initialization (called at start).

Returns:

0 on success, -1 on failure

This function is called when a programs starts. If there are any OS specific processing that is needed, it can be placed here. It is also acceptable to just return 0 if not special processing is needed.

Definition at line 72 of file os_none.c.

6.110.2.8 unsigned long os_random (void)

Get pseudo random value (not necessarily very strong).

Returns:

Pseudo random value

Definition at line 60 of file os_none.c.

6.110.2.9 char∗ os_readfile (const char ∗ *name*, size_t ∗ *len*)

Read a file to an allocated memory buffer.

Parameters:

name Name of the file to read

len For returning the length of the allocated buffer

Returns:

Pointer to the allocated buffer or NULL on failure

This function allocates memory and reads the given file to this buffer. Both binary and text files can be read with this function. The caller is responsible for freeing the returned buffer with os_free().

Definition at line 95 of file os_none.c.

6.110.2.10 char∗ os_rel2abs_path (const char ∗ *rel_path*)

Get an absolute path for a file.

Parameters:

rel_path Relative path to a file

Returns:

Absolute path for the file or NULL on failure

This function tries to convert a relative path of a file to an absolute path in order for the file to be found even if current working directory has changed. The returned value is allocated and caller is responsible for freeing it. It is acceptable to just return the same path in an allocated buffer, e.g., return strdup(rel_path). This function is only used to find configuration files when os_daemonize() may have changed the current working directory and relative path would be pointing to a different location.

Definition at line 66 of file os_none.c.

6.110.2.11 int os_setenv (const char ∗ *name*, const char ∗ *value*, int *overwrite*)

Set environment variable.

Parameters:

name Name of the variable

value Value to set to the variable

overwrite Whether existing variable should be overwritten

Returns:

0 on success, -1 on error

This function is only used for wpa_cli action scripts. OS wrapper does not need to implement this if such functionality is not needed.

Definition at line 83 of file os_none.c.

6.110.2.12 void os_sleep (os_time_t *sec*, os_time_t *usec*)

Sleep (sec, usec).

Parameters:

sec Number of seconds to sleep

usec Number of microseconds to sleep

Definition at line 25 of file os_none.c.

6.110.2.13 int os_unsetenv (const char ∗ *name*)

Delete environent variable.

Parameters:

name Name of the variable

Returns:

0 on success, -1 on error

This function is only used for wpa_cli action scripts. OS wrapper does not need to implement this if such functionality is not needed.

Definition at line 89 of file os_none.c.

6.110.2.14 void∗ os_zalloc (size_t *size*)

Allocate and zero memory.

Parameters:

size Number of bytes to allocate

Returns:

Pointer to allocated and zeroed memory or NULL on failure

Caller is responsible for freeing the returned buffer with os_free().

Definition at line 101 of file os_none.c.

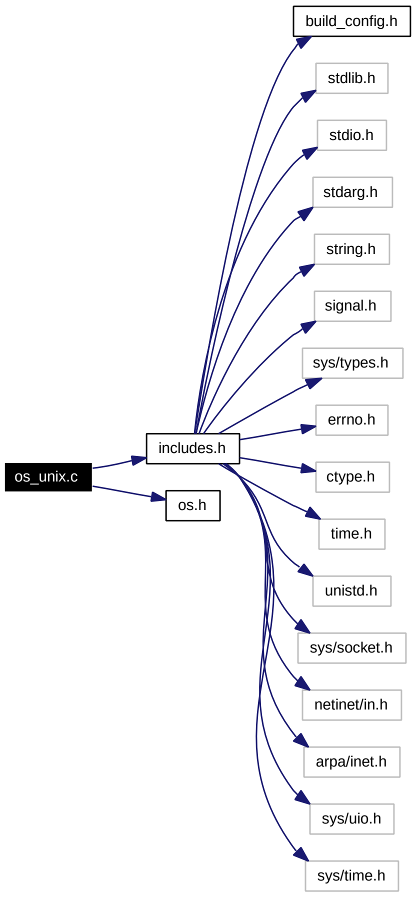
6.111 os_unix.c File Reference

wpa_supplicant/hostapd / OS specific functions for UNIX/POSIX systems

```
#include "includes.h"
```

```
#include "os.h"
```

Include dependency graph for os_unix.c:



Functions

- void os_sleep (os_time_t sec, os_time_t usec)

    Sleep (sec, usec).

- int os_get_time (struct os_time ∗t)

    Get current time (sec, usec).

- int os_mktime (int year, int month, int day, int hour, int min, int sec, os_time_t ∗t)

    Convert broken-down time into seconds since 1970-01-01.

- int os_daemonize (const char ∗pid_file)

    Run in the background (detach from the controlling terminal).

- void os_daemonize_terminate (const char ∗pid_file)

  Stop running in the background (remove pid file).

- int os_get_random (unsigned char ∗buf, size_t len)

  Get cryptographically strong pseudo random data.

- unsigned long os_random (void)

  Get pseudo random value (not necessarily very strong).

- char ∗ os_rel2abs_path (const char ∗rel_path)

  Get an absolute path for a file.

- int os_program_init (void)

  Program initialization (called at start).

- void os_program_deinit (void)

  Program deinitialization (called just before exit).

- int os_setenv (const char ∗name, const char ∗value, int overwrite)

  Set environment variable.

- int os_unsetenv (const char ∗name)

  Delete environent variable.

- char ∗ os_readfile (const char ∗name, size_t ∗len)

  Read a file to an allocated memory buffer.

- void ∗ os_zalloc (size_t size)

  Allocate and zero memory.

6.111.1 Detailed Description

wpa_supplicant/hostapd / OS specific functions for UNIX/POSIX systems

Copyright

Copyright (c) 2005-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file os_unix.c.

6.111.2 Function Documentation

6.111.2.1 int os_daemonize (const char ∗ *pid_file*)

Run in the background (detach from the controlling terminal).

Parameters:

pid_file File name to write the process ID to or NULL to skip this

Returns:

0 on success, -1 on failure

Definition at line 63 of file os_unix.c.

6.111.2.2 void os_daemonize_terminate (const char ∗ *pid_file*)

Stop running in the background (remove pid file).

Parameters:

pid_file File name to write the process ID to or NULL to skip this

Definition at line 82 of file os_unix.c.

6.111.2.3 int os_get_random (unsigned char ∗ *buf*, size_t *len*)

Get cryptographically strong pseudo random data.

Parameters:

buf Buffer for pseudo random data

len Length of the buffer

Returns:

0 on success, -1 on failure

Definition at line 89 of file os_unix.c.

6.111.2.4 int os_get_time (struct os_time ∗ *t*)

Get current time (sec, usec).

Parameters:

t Pointer to buffer for the time

Returns:

0 on success, -1 on failure

Definition at line 29 of file os_unix.c.

6.111.2.5 int os_mktime (int *year*, int *month*, int *day*, int *hour*, int *min*, int *sec*, os_time_t * *t*)

Convert broken-down time into seconds since 1970-01-01.

Parameters:

year Four digit year

month Month (1 .. 12)

day Day of month (1 .. 31)

hour Hour (0 .. 23)

min Minute (0 .. 59)

sec Second (0 .. 60)

t Buffer for returning calendar time representation (seconds since 1970-01-01 00:00:00)

Returns:

0 on success, -1 on failure

Definition at line 40 of file os_unix.c.

6.111.2.6 void os_program_deinit (void)

Program deinitialization (called just before exit).

This function is called just before a program exists. If there are any OS specific processing, e.g., freeing resourced allocated in os_program_init(), it should be done here. It is also acceptable for this function to do nothing.

Definition at line 162 of file os_unix.c.

6.111.2.7 int os_program_init (void)

Program initialization (called at start).

Returns:

0 on success, -1 on failure

This function is called when a programs starts. If there are any OS specific processing that is needed, it can be placed here. It is also acceptable to just return 0 if not special processing is needed.

Definition at line 156 of file os_unix.c.

6.111.2.8 unsigned long os_random (void)

Get pseudo random value (not necessarily very strong).

Returns:

Pseudo random value

Definition at line 107 of file os_unix.c.

6.111.2.9 char∗ os_readfile (const char ∗ *name*, size_t ∗ *len*)

Read a file to an allocated memory buffer.

Parameters:

name Name of the file to read

len For returning the length of the allocated buffer

Returns:

Pointer to the allocated buffer or NULL on failure

This function allocates memory and reads the given file to this buffer. Both binary and text files can be read with this function. The caller is responsible for freeing the returned buffer with os_free().

Definition at line 184 of file os_unix.c.

6.111.2.10 char∗ os_rel2abs_path (const char ∗ *rel_path*)

Get an absolute path for a file.

Parameters:

rel_path Relative path to a file

Returns:

Absolute path for the file or NULL on failure

This function tries to convert a relative path of a file to an absolute path in order for the file to be found even if current working directory has changed. The returned value is allocated and caller is responsible for freeing it. It is acceptable to just return the same path in an allocated buffer, e.g., return strdup(rel_path). This function is only used to find configuration files when os_daemonize() may have changed the current working directory and relative path would be pointing to a different location.

Definition at line 113 of file os_unix.c.

6.111.2.11 int os_setenv (const char ∗ *name*, const char ∗ *value*, int *overwrite*)

Set environment variable.

Parameters:

name Name of the variable

value Value to set to the variable

overwrite Whether existing variable should be overwritten

Returns:

0 on success, -1 on error

This function is only used for wpa_cli action scripts. OS wrapper does not need to implement this if such functionality is not needed.

Definition at line 167 of file os_unix.c.

6.111.2.12 void os_sleep (os_time_t *sec*, os_time_t *usec*)

Sleep (sec, usec).

Parameters:

sec Number of seconds to sleep

usec Number of microseconds to sleep

Definition at line 20 of file os_unix.c.

6.111.2.13 int os_unsetenv (const char ∗ *name*)

Delete environent variable.

Parameters:

name Name of the variable

Returns:

0 on success, -1 on error

This function is only used for wpa_cli action scripts. OS wrapper does not need to implement this if such functionality is not needed.

Definition at line 173 of file os_unix.c.

6.111.2.14 void∗ os_zalloc (size_t *size*)

Allocate and zero memory.

Parameters:

size Number of bytes to allocate

Returns:

Pointer to allocated and zeroed memory or NULL on failure

Caller is responsible for freeing the returned buffer with os_free().

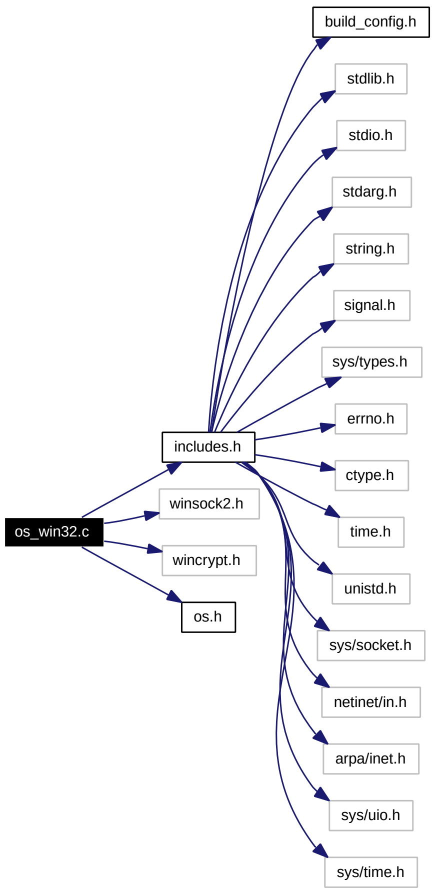
Definition at line 210 of file os_unix.c.

6.112 os_win32.c File Reference

wpa_supplicant/hostapd / OS specific functions for Win32 systems

```
#include "includes.h"
#include <winsock2.h>
#include <wincrypt.h>
#include "os.h"
```

Include dependency graph for os_win32.c:



Defines

- #define **EPOCHFILETIME** (116444736000000000ULL)

Functions

- void os_sleep (os_time_t sec, os_time_t usec)

  Sleep (sec, usec).

- int os_get_time (struct os_time ∗t)

Get current time (sec, usec).

- int os_mktime (int year, int month, int day, int hour, int min, int sec, os_time_t ∗t)
  Convert broken-down time into seconds since 1970-01-01.
- int os_daemonize (const char ∗pid_file)
  Run in the background (detach from the controlling terminal).
- void os_daemonize_terminate (const char ∗pid_file)
  Stop running in the background (remove pid file).
- int os_get_random (unsigned char ∗buf, size_t len)
  Get cryptographically strong pseudo random data.
- unsigned long os_random (void)
  Get pseudo random value (not necessarily very strong).
- char ∗ os_rel2abs_path (const char ∗rel_path)
  Get an absolute path for a file.
- int os_program_init (void)
  Program initialization (called at start).
- void os_program_deinit (void)
  Program deinitialization (called just before exit).
- int os_setenv (const char ∗name, const char ∗value, int overwrite)
  Set environment variable.
- int os_unsetenv (const char ∗name)
  Delete environent variable.
- char ∗ os_readfile (const char ∗name, size_t ∗len)
  Read a file to an allocated memory buffer.
- void ∗ os_zalloc (size_t size)
  Allocate and zero memory.

6.112.1 Detailed Description

wpa_supplicant/hostapd / OS specific functions for Win32 systems

Copyright
Copyright (c) 2005-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file os_win32.c.

6.112.2 Function Documentation

6.112.2.1 int os_daemonize (const char ∗ *pid_file*)

Run in the background (detach from the controlling terminal).

Parameters:

pid_file File name to write the process ID to or NULL to skip this

Returns:

0 on success, -1 on failure

Definition at line 79 of file os_win32.c.

6.112.2.2 void os_daemonize_terminate (const char ∗ *pid_file*)

Stop running in the background (remove pid file).

Parameters:

pid_file File name to write the process ID to or NULL to skip this

Definition at line 86 of file os_win32.c.

6.112.2.3 int os_get_random (unsigned char ∗ *buf*, size_t *len*)

Get cryptographically strong pseudo random data.

Parameters:

buf Buffer for pseudo random data

len Length of the buffer

Returns:

0 on success, -1 on failure

Definition at line 91 of file os_win32.c.

6.112.2.4 int os_get_time (struct os_time ∗ *t*)

Get current time (sec, usec).

Parameters:

t Pointer to buffer for the time

Returns:

0 on success, -1 on failure

Definition at line 31 of file os_win32.c.

6.112.2.5 int os_mktime (int *year*, int *month*, int *day*, int *hour*, int *min*, int *sec*, os_time_t * *t*)

Convert broken-down time into seconds since 1970-01-01.

Parameters:

year Four digit year

month Month (1 .. 12)

day Day of month (1 .. 31)

hour Hour (0 .. 23)

min Minute (0 .. 59)

sec Second (0 .. 60)

t Buffer for returning calendar time representation (seconds since 1970-01-01 00:00:00)

Returns:

0 on success, -1 on failure

Definition at line 56 of file os_win32.c.

6.112.2.6 void os_program_deinit (void)

Program deinitialization (called just before exit).

This function is called just before a program exists. If there are any OS specific processing, e.g., freeing resourced allocated in os_program_init(), it should be done here. It is also acceptable for this function to do nothing.

Definition at line 132 of file os_win32.c.

6.112.2.7 int os_program_init (void)

Program initialization (called at start).

Returns:

0 on success, -1 on failure

This function is called when a programs starts. If there are any OS specific processing that is needed, it can be placed here. It is also acceptable to just return 0 if not special processing is needed.

Definition at line 119 of file os_win32.c.

6.112.2.8 unsigned long os_random (void)

Get pseudo random value (not necessarily very strong).

Returns:

Pseudo random value

Definition at line 107 of file os_win32.c.

6.112.2.9 char∗ os_readfile (const char ∗ *name*, size_t ∗ *len*)

Read a file to an allocated memory buffer.

Parameters:

name Name of the file to read

len For returning the length of the allocated buffer

Returns:

Pointer to the allocated buffer or NULL on failure

This function allocates memory and reads the given file to this buffer. Both binary and text files can be read with this function. The caller is responsible for freeing the returned buffer with os_free().

Definition at line 152 of file os_win32.c.

6.112.2.10 char∗ os_rel2abs_path (const char ∗ *rel_path*)

Get an absolute path for a file.

Parameters:

rel_path Relative path to a file

Returns:

Absolute path for the file or NULL on failure

This function tries to convert a relative path of a file to an absolute path in order for the file to be found even if current working directory has changed. The returned value is allocated and caller is responsible for freeing it. It is acceptable to just return the same path in an allocated buffer, e.g., return strdup(rel_path). This function is only used to find configuration files when os_daemonize() may have changed the current working directory and relative path would be pointing to a different location.

Definition at line 113 of file os_win32.c.

6.112.2.11 int os_setenv (const char ∗ *name*, const char ∗ *value*, int *overwrite*)

Set environment variable.

Parameters:

name Name of the variable

value Value to set to the variable

overwrite Whether existing variable should be overwritten

Returns:

0 on success, -1 on error

This function is only used for wpa_cli action scripts. OS wrapper does not need to implement this if such functionality is not needed.

Definition at line 140 of file os_win32.c.

6.112.2.12 void os_sleep (os_time_t *sec*, os_time_t *usec*)

Sleep (sec, usec).

Parameters:

sec Number of seconds to sleep

usec Number of microseconds to sleep

Definition at line 22 of file os_win32.c.

6.112.2.13 int os_unsetenv (const char ∗ *name*)

Delete environent variable.

Parameters:

name Name of the variable

Returns:

0 on success, -1 on error

This function is only used for wpa_cli action scripts. OS wrapper does not need to implement this if such functionality is not needed.

Definition at line 146 of file os_win32.c.

6.112.2.14 void∗ os_zalloc (size_t *size*)

Allocate and zero memory.

Parameters:

size Number of bytes to allocate

Returns:

Pointer to allocated and zeroed memory or NULL on failure

Caller is responsible for freeing the returned buffer with os_free().

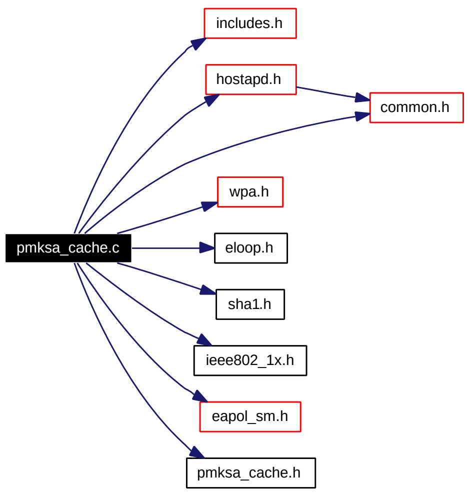
Definition at line 178 of file os_win32.c.

6.113 pmksa_cache.c File Reference

hostapd - PMKSA cache for IEEE 802.11i RSN

```
#include "includes.h"
#include "hostapd.h"
#include "common.h"
#include "wpa.h"
#include "eloop.h"
#include "sha1.h"
#include "ieee802_1x.h"
#include "eapol_sm.h"
#include "pmksa_cache.h"
```

Include dependency graph for pmksa_cache.c:



Defines

- #define **PMKID_HASH_SIZE** 128
- #define **PMKID_HASH**(pmkid) (unsigned int) ((pmkid)[0] & 0x7f)

Functions

- void rsn_pmkid (const u8 ∗pmk, size_t pmk_len, const u8 ∗aa, const u8 ∗spa, u8 ∗pmkid)

  Calculate PMK identifier.

- void **pmksa_cache_to_eapol_data** (struct rsn_pmksa_cache_entry ∗entry, struct eapol_state_-machine ∗eapol)
- rsn_pmksa_cache_entry ∗ pmksa_cache_add (struct rsn_pmksa_cache ∗pmksa, const u8 ∗pmk, size_t pmk_len, const u8 ∗aa, const u8 ∗spa, int session_timeout, struct eapol_state_machine ∗eapol)

Add a PMKSA cache entry.

- void pmksa_cache_deinit (struct rsn_pmksa_cache ∗pmksa)

  Free all entries in PMKSA cache.

- rsn_pmksa_cache_entry ∗ pmksa_cache_get (struct rsn_pmksa_cache ∗pmksa, const u8 ∗spa, const u8 ∗pmkid)

  Fetch a PMKSA cache entry.

- rsn_pmksa_cache ∗ pmksa_cache_init (void(∗free_cb)(struct rsn_pmksa_cache_entry ∗entry, void ∗ctx), void ∗ctx)

  Initialize PMKSA cache.

6.113.1 Detailed Description

hostapd - PMKSA cache for IEEE 802.11i RSN

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file pmksa_cache.c.

6.113.2 Function Documentation

6.113.2.1 struct rsn_pmksa_cache_entry∗ pmksa_cache_add (struct rsn_pmksa_cache ∗ *pmksa*, const u8 ∗ *pmk*, size_t *pmk_len*, const u8 ∗ *aa*, const u8 ∗ *spa*, int *session_timeout*, struct eapol_state_machine ∗ *eapol*)

Add a PMKSA cache entry.

Parameters:

pmksa Pointer to PMKSA cache data from pmksa_cache_init()

pmk The new pairwise master key

pmk_len PMK length in bytes, usually PMK_LEN (32)

aa Authenticator address

spa Supplicant address

session_timeout Session timeout

eapol Pointer to EAPOL state machine data

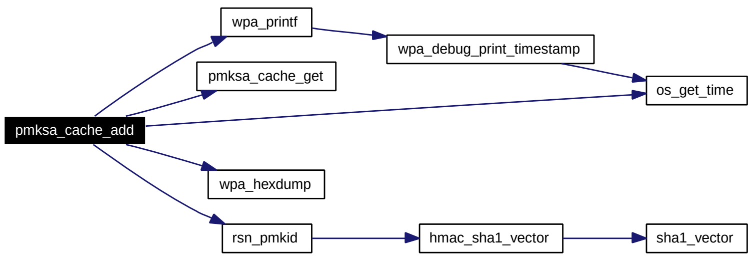
Returns:

Pointer to the added PMKSA cache entry or NULL on error

This function create a PMKSA entry for a new PMK and adds it to the PMKSA cache. If an old entry is already in the cache for the same Supplicant, this entry will be replaced with the new entry. PMKID will be calculated based on the PMK.

Definition at line 230 of file pmksa_cache.c.

Here is the call graph for this function:



6.113.2.2 void pmksa_cache_deinit (struct rsn_pmksa_cache ∗ *pmksa*)

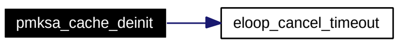
Free all entries in PMKSA cache.

Parameters:

pmksa Pointer to PMKSA cache data from pmksa_cache_init()

Definition at line 303 of file pmksa_cache.c.

Here is the call graph for this function:



6.113.2.3 struct rsn_pmksa_cache_entry∗ pmksa_cache_get (struct rsn_pmksa_cache ∗ *pmksa*, const u8 ∗ *spa*, const u8 ∗ *pmkid*)

Fetch a PMKSA cache entry.

Parameters:

pmksa Pointer to PMKSA cache data from pmksa_cache_init()

spa Supplicant address or NULL to match any

pmkid PMKID or NULL to match any

Returns:

Pointer to PMKSA cache entry or NULL if no match was found

Definition at line 332 of file pmksa_cache.c.

6.113.2.4 struct rsn_pmksa_cache∗ pmksa_cache_init (void(∗)(struct rsn_pmksa_cache_entry ∗entry, void ∗ctx) *free_cb*, void ∗ *ctx*)

Initialize PMKSA cache.

Parameters:

free_cb Callback function to be called when a PMKSA cache entry is freed

ctx Context pointer for free_cb function

Returns:

Pointer to PMKSA cache data or NULL on failure

Definition at line 360 of file pmksa_cache.c.

6.113.2.5 void rsn_pmkid (const u8 ∗ *pmk*, size_t *pmk_len*, const u8 ∗ *aa*, const u8 ∗ *spa*, u8 ∗ *pmkid*)

Calculate PMK identifier.

Parameters:

pmk Pairwise master key

pmk_len Length of pmk in bytes

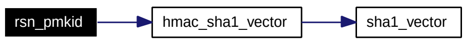
aa Authenticator address

spa Supplicant address

IEEE Std 802.11i-2004 - 8.5.1.2 Pairwise key hierarchy PMKID = HMAC-SHA1-128(PMK, "PMK Name" || AA || SPA)

Definition at line 54 of file pmksa_cache.c.

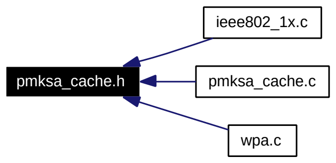
Here is the call graph for this function:



6.114 pmksa_cache.h File Reference

hostapd - PMKSA cache for IEEE 802.11i RSN

This graph shows which files directly or indirectly include this file:



Functions

- rsn_pmksa_cache ∗ pmksa_cache_init (void(∗free_cb)(struct rsn_pmksa_cache_entry ∗entry, void ∗ctx), void ∗ctx)

  Initialize PMKSA cache.

- void pmksa_cache_deinit (struct rsn_pmksa_cache ∗pmksa)

  Free all entries in PMKSA cache.

- rsn_pmksa_cache_entry ∗ pmksa_cache_get (struct rsn_pmksa_cache ∗pmksa, const u8 ∗spa, const u8 ∗pmkid)

  Fetch a PMKSA cache entry.

- rsn_pmksa_cache_entry ∗ pmksa_cache_add (struct rsn_pmksa_cache ∗pmksa, const u8 ∗pmk, size_t pmk_len, const u8 ∗aa, const u8 ∗spa, int session_timeout, struct eapol_state_machine ∗eapol)

  Add a PMKSA cache entry.

- void **pmksa_cache_to_eapol_data** (struct rsn_pmksa_cache_entry ∗entry, struct eapol_state_-machine ∗eapol)
- void rsn_pmkid (const u8 ∗pmk, size_t pmk_len, const u8 ∗aa, const u8 ∗spa, u8 ∗pmkid)

  Calculate PMK identifier.

6.114.1 Detailed Description

hostapd - PMKSA cache for IEEE 802.11i RSN

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file pmksa_cache.h.

6.114.2 Function Documentation

6.114.2.1 struct rsn_pmksa_cache_entry∗ pmksa_cache_add (struct rsn_pmksa_cache ∗ *pmksa*, const u8 ∗ *pmk*, size_t *pmk_len*, const u8 ∗ *aa*, const u8 ∗ *spa*, int *session_timeout*, struct eapol_state_machine ∗ *eapol*)

Add a PMKSA cache entry.

Parameters:

pmksa Pointer to PMKSA cache data from pmksa_cache_init()

pmk The new pairwise master key

pmk_len PMK length in bytes, usually PMK_LEN (32)

aa Authenticator address

spa Supplicant address

session_timeout Session timeout

eapol Pointer to EAPOL state machine data

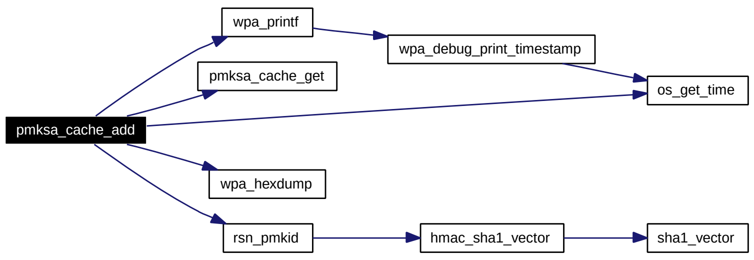
Returns:

Pointer to the added PMKSA cache entry or NULL on error

This function create a PMKSA entry for a new PMK and adds it to the PMKSA cache. If an old entry is already in the cache for the same Supplicant, this entry will be replaced with the new entry. PMKID will be calculated based on the PMK.

Definition at line 230 of file pmksa_cache.c.

Here is the call graph for this function:



6.114.2.2 void pmksa_cache_deinit (struct rsn_pmksa_cache ∗ *pmksa*)

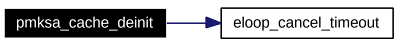
Free all entries in PMKSA cache.

Parameters:

pmksa Pointer to PMKSA cache data from pmksa_cache_init()

Definition at line 303 of file pmksa_cache.c.

Here is the call graph for this function:



6.114.2.3 struct rsn_pmksa_cache_entry∗ pmksa_cache_get (struct rsn_pmksa_cache ∗ *pmksa*, const u8 ∗ *spa*, const u8 ∗ *pmkid*)

Fetch a PMKSA cache entry.

Parameters:

pmksa Pointer to PMKSA cache data from pmksa_cache_init()

spa Supplicant address or NULL to match any

pmkid PMKID or NULL to match any

Returns:

Pointer to PMKSA cache entry or NULL if no match was found

Definition at line 332 of file pmksa_cache.c.

6.114.2.4 struct rsn_pmksa_cache∗ pmksa_cache_init (void(∗)(struct rsn_pmksa_cache_entry ∗entry, void ∗ctx) *free_cb*, void ∗ *ctx*)

Initialize PMKSA cache.

Parameters:

free_cb Callback function to be called when a PMKSA cache entry is freed

ctx Context pointer for free_cb function

Returns:

Pointer to PMKSA cache data or NULL on failure

Definition at line 360 of file pmksa_cache.c.

6.114.2.5 void rsn_pmkid (const u8 ∗ *pmk*, size_t *pmk_len*, const u8 ∗ *aa*, const u8 ∗ *spa*, u8 ∗ *pmkid*)

Calculate PMK identifier.

Parameters:

pmk Pairwise master key

pmk_len Length of pmk in bytes

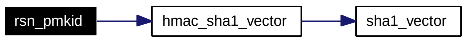
aa Authenticator address

spa Supplicant address

IEEE Std 802.11i-2004 - 8.5.1.2 Pairwise key hierarchy PMKID = HMAC-SHA1-128(PMK, "PMK Name" || AA || SPA)

Definition at line 54 of file pmksa_cache.c.

Here is the call graph for this function:

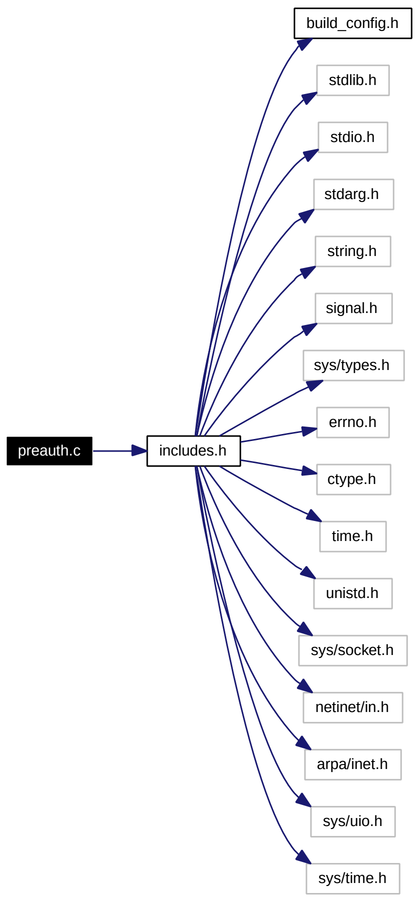


6.115 preauth.c File Reference

hostapd - Authenticator for IEEE 802.11i RSN pre-authentication

```
#include "includes.h"
```

Include dependency graph for preauth.c:



6.115.1 Detailed Description

hostapd - Authenticator for IEEE 802.11i RSN pre-authentication

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

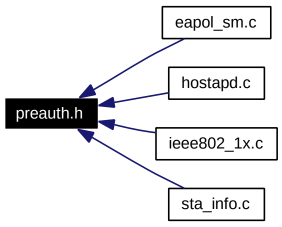
See README and COPYING for more details.

Definition in file preauth.c.

6.116 preauth.h File Reference

hostapd - Authenticator for IEEE 802.11i RSN pre-authentication

This graph shows which files directly or indirectly include this file:



6.116.1 Detailed Description

hostapd - Authenticator for IEEE 802.11i RSN pre-authentication

Copyright

Definition in file preauth.h.

6.117 radius.c File Reference

hostapd / RADIUS message processing

`#include "includes.h"`

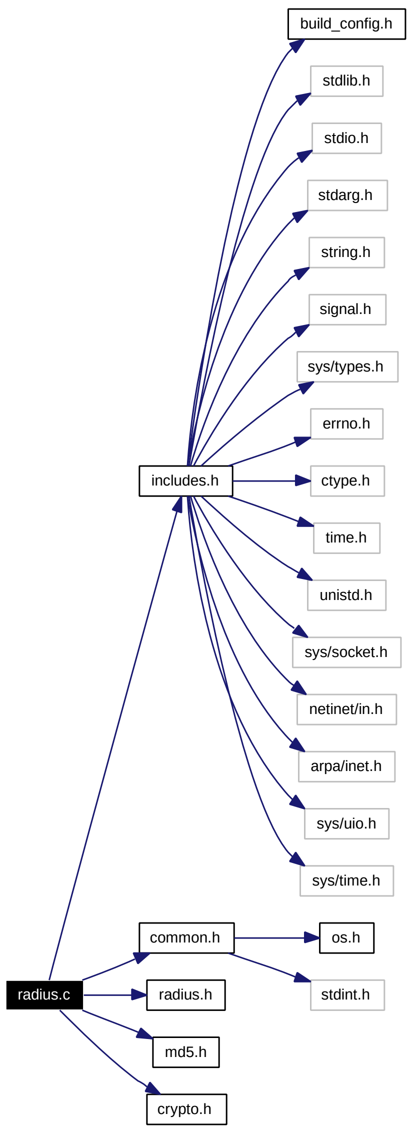
`#include "common.h"`

`#include "radius.h"`

`#include "md5.h"`

`#include "crypto.h"`

Include dependency graph for radius.c:



Defines

- #define **RADIUS_ATTRS** (sizeof(radius_attrs) / sizeof(radius_attrs[0]))

Functions

- radius_msg ∗ **radius_msg_new** (u8 code, u8 identifier)
- int **radius_msg_initialize** (struct radius_msg ∗msg, size_t init_len)
- void **radius_msg_set_hdr** (struct radius_msg ∗msg, u8 code, u8 identifier)
- void **radius_msg_free** (struct radius_msg ∗msg)
- void **radius_msg_dump** (struct radius_msg ∗msg)
- int **radius_msg_finish** (struct radius_msg ∗msg, u8 ∗secret, size_t secret_len)
- int **radius_msg_finish_srv** (struct radius_msg ∗msg, const u8 ∗secret, size_t secret_len, const u8 ∗req_authenticator)
- void **radius_msg_finish_acct** (struct radius_msg ∗msg, u8 ∗secret, size_t secret_len)
- radius_attr_hdr ∗ **radius_msg_add_attr** (struct radius_msg ∗msg, u8 type, const u8 ∗data, size_t data_len)
- radius_msg ∗ **radius_msg_parse** (const u8 ∗data, size_t len)
- int **radius_msg_add_eap** (struct radius_msg ∗msg, const u8 ∗data, size_t data_len)
- u8 ∗ **radius_msg_get_eap** (struct radius_msg ∗msg, size_t ∗eap_len)
- int **radius_msg_verify_msg_auth** (struct radius_msg ∗msg, const u8 ∗secret, size_t secret_len, const u8 ∗req_auth)
- int **radius_msg_verify** (struct radius_msg ∗msg, const u8 ∗secret, size_t secret_len, struct radius_-msg ∗sent_msg, int auth)
- int **radius_msg_copy_attr** (struct radius_msg ∗dst, struct radius_msg ∗src, u8 type)
- void **radius_msg_make_authenticator** (struct radius_msg ∗msg, const u8 ∗data, size_t len)
- radius_ms_mppe_keys ∗ **radius_msg_get_ms_keys** (struct radius_msg ∗msg, struct radius_msg ∗sent_msg, u8 ∗secret, size_t secret_len)
- radius_ms_mppe_keys ∗ **radius_msg_get_cisco_keys** (struct radius_msg ∗msg, struct radius_msg ∗sent_msg, u8 ∗secret, size_t secret_len)
- int **radius_msg_add_mppe_keys** (struct radius_msg ∗msg, const u8 ∗req_authenticator, const u8 ∗secret, size_t secret_len, const u8 ∗send_key, size_t send_key_len, const u8 ∗recv_key, size_t recv_key_len)
- radius_attr_hdr ∗ **radius_msg_add_attr_user_password** (struct radius_msg ∗msg, u8 ∗data, size_t data_len, u8 ∗secret, size_t secret_len)
- int **radius_msg_get_attr** (struct radius_msg ∗msg, u8 type, u8 ∗buf, size_t len)
- int **radius_msg_get_attr_ptr** (struct radius_msg ∗msg, u8 type, u8 ∗∗buf, size_t ∗len, const u8 ∗start)
- int **radius_msg_count_attr** (struct radius_msg ∗msg, u8 type, int min_len)
- int radius_msg_get_vlanid (struct radius_msg ∗msg)

    Parse RADIUS attributes for VLAN tunnel information.

6.117.1 Detailed Description

hostapd / RADIUS message processing

Copyright

Copyright (c) 2002-2005, Jouni Malinen <jkmaline@cc.hut.fi>

Definition in file radius.c.

6.117.2 Function Documentation

6.117.2.1 int radius_msg_get_vlanid (struct radius_msg ∗ *msg*)

Parse RADIUS attributes for VLAN tunnel information.

Parameters:

msg RADIUS message

Returns:

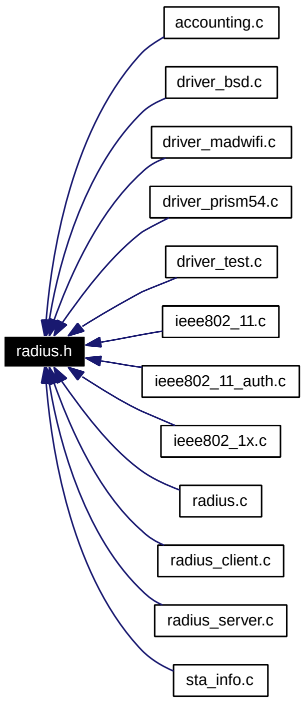
VLAN ID for the first tunnel configuration of -1 if none is found

Definition at line 1170 of file radius.c.

6.118 radius.h File Reference

hostapd / RADIUS message processing

This graph shows which files directly or indirectly include this file:



Defines

- #define **RADIUS_MAX_ATTR_LEN** (255 - sizeof(struct radius_attr_hdr))
- #define **RADIUS_TERMINATION_ACTION_DEFAULT** 0
- #define **RADIUS_TERMINATION_ACTION_RADIUS_REQUEST** 1
- #define **RADIUS_NAS_PORT_TYPE_IEEE_802_11** 19
- #define **RADIUS_ACCT_STATUS_TYPE_START** 1
- #define **RADIUS_ACCT_STATUS_TYPE_STOP** 2
- #define **RADIUS_ACCT_STATUS_TYPE_INTERIM_UPDATE** 3
- #define **RADIUS_ACCT_STATUS_TYPE_ACCOUNTING_ON** 7
- #define **RADIUS_ACCT_STATUS_TYPE_ACCOUNTING_OFF** 8
- #define **RADIUS_ACCT_AUTHENTIC_RADIUS** 1
- #define **RADIUS_ACCT_AUTHENTIC_LOCAL** 2
- #define **RADIUS_ACCT_AUTHENTIC_REMOTE** 3
- #define **RADIUS_ACCT_TERMINATE_CAUSE_USER_REQUEST** 1
- #define **RADIUS_ACCT_TERMINATE_CAUSE_LOST_CARRIER** 2
- #define **RADIUS_ACCT_TERMINATE_CAUSE_LOST_SERVICE** 3
- #define **RADIUS_ACCT_TERMINATE_CAUSE_IDLE_TIMEOUT** 4
- #define **RADIUS_ACCT_TERMINATE_CAUSE_SESSION_TIMEOUT** 5
- #define **RADIUS_ACCT_TERMINATE_CAUSE_ADMIN_RESET** 6
- #define **RADIUS_ACCT_TERMINATE_CAUSE_ADMIN_REBOOT** 7
- #define **RADIUS_ACCT_TERMINATE_CAUSE_PORT_ERROR** 8

- #define **RADIUS_ACCT_TERMINATE_CAUSE_NAS_ERROR** 9
- #define **RADIUS_ACCT_TERMINATE_CAUSE_NAS_REQUEST** 10
- #define **RADIUS_ACCT_TERMINATE_CAUSE_NAS_REBOOT** 11
- #define **RADIUS_ACCT_TERMINATE_CAUSE_PORT_UNNEEDED** 12
- #define **RADIUS_ACCT_TERMINATE_CAUSE_PORT_PREEMPTED** 13
- #define **RADIUS_ACCT_TERMINATE_CAUSE_PORT_SUSPENDED** 14
- #define **RADIUS_ACCT_TERMINATE_CAUSE_SERVICE_UNAVAILABLE** 15
- #define **RADIUS_ACCT_TERMINATE_CAUSE_CALLBACK** 16
- #define **RADIUS_ACCT_TERMINATE_CAUSE_USER_ERROR** 17
- #define **RADIUS_ACCT_TERMINATE_CAUSE_HOST_REQUEST** 18
- #define **RADIUS_TUNNEL_TAGS** 32
- #define **RADIUS_TUNNEL_TYPE_PPTP** 1
- #define **RADIUS_TUNNEL_TYPE_L2TP** 3
- #define **RADIUS_TUNNEL_TYPE_IPIP** 7
- #define **RADIUS_TUNNEL_TYPE_GRE** 10
- #define **RADIUS_TUNNEL_TYPE_VLAN** 13
- #define **RADIUS_TUNNEL_MEDIUM_TYPE_IPV4** 1
- #define **RADIUS_TUNNEL_MEDIUM_TYPE_IPV6** 2
- #define **RADIUS_TUNNEL_MEDIUM_TYPE_802** 6
- #define **RADIUS_VENDOR_ID_CISCO** 9
- #define **RADIUS_CISCO_AV_PAIR** 1
- #define **RADIUS_VENDOR_ID_MICROSOFT** 311
- #define **RADIUS_DEFAULT_MSG_SIZE** 1024
- #define **RADIUS_DEFAULT_ATTR_COUNT** 16
- #define **RADIUS_802_1X_ADDR_FORMAT** "%02X-%02X-%02X-%02X-%02X-%02X"
- #define **RADIUS_ADDR_FORMAT** "%02x%02x%02x%02x%02x%02x"

Enumerations

- enum {

  RADIUS_CODE_ACCESS_REQUEST = 1, **RADIUS_CODE_ACCESS_ACCEPT** = 2, **RADIUS_CODE_ACCESS_REJECT** = 3, **RADIUS_CODE_ACCOUNTING_REQUEST** = 4,

  RADIUS_CODE_ACCOUNTING_RESPONSE = 5, **RADIUS_CODE_ACCESS_-CHALLENGE** = 11, **RADIUS_CODE_STATUS_SERVER** = 12, **RADIUS_CODE_STATUS_-CLIENT** = 13,

  RADIUS_CODE_RESERVED = 255 }
- enum {

  RADIUS_ATTR_USER_NAME = 1, **RADIUS_ATTR_USER_PASSWORD** = 2, **RADIUS_-ATTR_NAS_IP_ADDRESS** = 4, **RADIUS_ATTR_NAS_PORT** = 5,

  RADIUS_ATTR_FRAMED_MTU = 12, **RADIUS_ATTR_STATE** = 24, **RADIUS_ATTR_-CLASS** = 25, **RADIUS_ATTR_VENDOR_SPECIFIC** = 26,

  RADIUS_ATTR_SESSION_TIMEOUT = 27, **RADIUS_ATTR_IDLE_TIMEOUT** = 28, **RADIUS_ATTR_TERMINATION_ACTION** = 29, **RADIUS_ATTR_CALLED_STATION_ID** = 30,

  RADIUS_ATTR_CALLING_STATION_ID = 31, **RADIUS_ATTR_NAS_IDENTIFIER** = 32, **RADIUS_ATTR_ACCT_STATUS_TYPE** = 40, **RADIUS_ATTR_ACCT_DELAY_TIME** = 41,

  RADIUS_ATTR_ACCT_INPUT_OCTETS = 42, **RADIUS_ATTR_ACCT_OUTPUT_-OCTETS** = 43, **RADIUS_ATTR_ACCT_SESSION_ID** = 44, **RADIUS_ATTR_ACCT_-AUTHENTIC** = 45,

RADIUS_ATTR_ACCT_SESSION_TIME = 46, **RADIUS_ATTR_ACCT_INPUT_PACKETS** = 47, **RADIUS_ATTR_ACCT_OUTPUT_PACKETS** = 48, **RADIUS_ATTR_ACCT_-TERMINATE_CAUSE** = 49,

RADIUS_ATTR_ACCT_MULTI_SESSION_ID = 50, **RADIUS_ATTR_ACCT_LINK_-COUNT** = 51, **RADIUS_ATTR_ACCT_INPUT_GIGAWORDS** = 52, **RADIUS_ATTR_-ACCT_OUTPUT_GIGAWORDS** = 53,

RADIUS_ATTR_EVENT_TIMESTAMP = 55, **RADIUS_ATTR_NAS_PORT_TYPE** = 61, **RADIUS_ATTR_TUNNEL_TYPE** = 64, **RADIUS_ATTR_TUNNEL_MEDIUM_TYPE** = 65,

RADIUS_ATTR_CONNECT_INFO = 77, **RADIUS_ATTR_EAP_MESSAGE** = 79, **RADIUS_-ATTR_MESSAGE_AUTHENTICATOR** = 80, **RADIUS_ATTR_TUNNEL_PRIVATE_-GROUP_ID** = 81,

RADIUS_ATTR_ACCT_INTERIM_INTERVAL = 85, **RADIUS_ATTR_NAS_IPV6_-ADDRESS** = 95 }

- enum { **RADIUS_VENDOR_ATTR_MS_MPPE_SEND_KEY** = 16, **RADIUS_VENDOR_-ATTR_MS_MPPE_RECV_KEY** = 17 }

Functions

- radius_msg ∗ **radius_msg_new** (u8 code, u8 identifier)
- int **radius_msg_initialize** (struct radius_msg ∗msg, size_t init_len)
- void **radius_msg_set_hdr** (struct radius_msg ∗msg, u8 code, u8 identifier)
- void **radius_msg_free** (struct radius_msg ∗msg)
- void **radius_msg_dump** (struct radius_msg ∗msg)
- int **radius_msg_finish** (struct radius_msg ∗msg, u8 ∗secret, size_t secret_len)
- int **radius_msg_finish_srv** (struct radius_msg ∗msg, const u8 ∗secret, size_t secret_len, const u8 ∗req_authenticator)
- void **radius_msg_finish_acct** (struct radius_msg ∗msg, u8 ∗secret, size_t secret_len)
- radius_attr_hdr ∗ **radius_msg_add_attr** (struct radius_msg ∗msg, u8 type, const u8 ∗data, size_t data_len)
- radius_msg ∗ **radius_msg_parse** (const u8 ∗data, size_t len)
- int **radius_msg_add_eap** (struct radius_msg ∗msg, const u8 ∗data, size_t data_len)
- u8 ∗ **radius_msg_get_eap** (struct radius_msg ∗msg, size_t ∗len)
- int **radius_msg_verify** (struct radius_msg ∗msg, const u8 ∗secret, size_t secret_len, struct radius_-msg ∗sent_msg, int auth)
- int **radius_msg_verify_msg_auth** (struct radius_msg ∗msg, const u8 ∗secret, size_t secret_len, const u8 ∗req_auth)
- int **radius_msg_copy_attr** (struct radius_msg ∗dst, struct radius_msg ∗src, u8 type)
- void **radius_msg_make_authenticator** (struct radius_msg ∗msg, const u8 ∗data, size_t len)
- radius_ms_mppe_keys ∗ **radius_msg_get_ms_keys** (struct radius_msg ∗msg, struct radius_msg ∗sent_msg, u8 ∗secret, size_t secret_len)
- radius_ms_mppe_keys ∗ **radius_msg_get_cisco_keys** (struct radius_msg ∗msg, struct radius_msg ∗sent_msg, u8 ∗secret, size_t secret_len)
- int **radius_msg_add_mppe_keys** (struct radius_msg ∗msg, const u8 ∗req_authenticator, const u8 ∗secret, size_t secret_len, const u8 ∗send_key, size_t send_key_len, const u8 ∗recv_key, size_t recv_key_len)
- radius_attr_hdr ∗ **radius_msg_add_attr_user_password** (struct radius_msg ∗msg, u8 ∗data, size_t data_len, u8 ∗secret, size_t secret_len)
- int **radius_msg_get_attr** (struct radius_msg ∗msg, u8 type, u8 ∗buf, size_t len)
- int radius_msg_get_vlanid (struct radius_msg ∗msg)

  Parse RADIUS attributes for VLAN tunnel information.

- int **radius_msg_get_attr_ptr** (struct radius_msg ∗msg, u8 type, u8 ∗∗buf, size_t ∗len, const u8 ∗start)
- int **radius_msg_count_attr** (struct radius_msg ∗msg, u8 type, int min_len)

Variables

- radius_hdr **STRUCT_PACKED**

6.118.1 Detailed Description

hostapd / RADIUS message processing

Copyright

Copyright (c) 2002-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file radius.h.

6.118.2 Function Documentation

6.118.2.1 int radius_msg_get_vlanid (struct radius_msg ∗ *msg*)

Parse RADIUS attributes for VLAN tunnel information.

Parameters:

msg RADIUS message

Returns:

VLAN ID for the first tunnel configuration of -1 if none is found

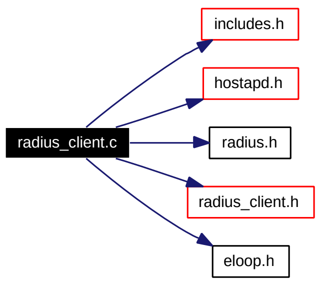
Definition at line 1170 of file radius.c.

6.119 radius_client.c File Reference

hostapd / RADIUS client

```
#include "includes.h"
#include "hostapd.h"
#include "radius.h"
#include "radius_client.h"
#include "eloop.h"
```

Include dependency graph for radius_client.c:



Defines

- #define **RADIUS_CLIENT_FIRST_WAIT** 3
- #define **RADIUS_CLIENT_MAX_WAIT** 120
- #define **RADIUS_CLIENT_MAX_RETRIES** 10
- #define **RADIUS_CLIENT_MAX_ENTRIES** 30
- #define **RADIUS_CLIENT_NUM_FAILOVER** 4

Functions

- int **radius_client_register** (struct radius_client_data ∗radius, RadiusType msg_type, RadiusRx-Result(∗handler)(struct radius_msg ∗msg, struct radius_msg ∗req, u8 ∗shared_secret, size_t shared_-secret_len, void ∗data), void ∗data)
- int **radius_client_send** (struct radius_client_data ∗radius, struct radius_msg ∗msg, RadiusType msg_type, const u8 ∗addr)
- u8 **radius_client_get_id** (struct radius_client_data ∗radius)
- void **radius_client_flush** (struct radius_client_data ∗radius, int only_auth)
- void **radius_client_update_acct_msgs** (struct radius_client_data ∗radius, u8 ∗shared_secret, size_t shared_secret_len)
- radius_client_data ∗ **radius_client_init** (void ∗ctx, struct hostapd_radius_servers ∗conf)
- void **radius_client_deinit** (struct radius_client_data ∗radius)
- void **radius_client_flush_auth** (struct radius_client_data ∗radius, u8 ∗addr)
- int **radius_client_get_mib** (struct radius_client_data ∗radius, char ∗buf, size_t buflen)
- radius_client_data ∗ **radius_client_reconfig** (struct radius_client_data ∗old, void ∗ctx, struct hostapd_radius_servers ∗oldconf, struct hostapd_radius_servers ∗newconf)

6.119.1 Detailed Description

hostapd / RADIUS client

Copyright

Copyright (c) 2002-2005, Jouni Malinen <jkmaline@cc.hut.fi>

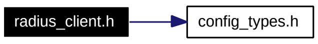
Definition in file radius_client.c.

6.120 radius_client.h File Reference

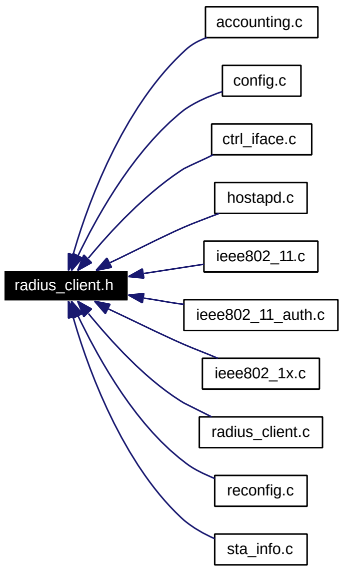
hostapd / RADIUS client

```
#include "config_types.h"
```

Include dependency graph for radius_client.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum **RadiusType** { **RADIUS_AUTH**, **RADIUS_ACCT**, **RADIUS_ACCT_INTERIM** }
- enum **RadiusRxResult** { **RADIUS_RX_PROCESSED**, **RADIUS_RX_QUEUED**, **RADIUS_-RX_UNKNOWN**, **RADIUS_RX_INVALID_AUTHENTICATOR** }

Functions

- int **radius_client_register** (struct radius_client_data ∗radius, RadiusType msg_type, RadiusRx-Result(∗handler)(struct radius_msg ∗msg, struct radius_msg ∗req, u8 ∗shared_secret, size_t shared_-secret_len, void ∗data), void ∗data)
- int **radius_client_send** (struct radius_client_data ∗radius, struct radius_msg ∗msg, RadiusType msg_type, const u8 ∗addr)
- u8 **radius_client_get_id** (struct radius_client_data ∗radius)
- void **radius_client_flush** (struct radius_client_data ∗radius, int only_auth)
- radius_client_data ∗ **radius_client_init** (void ∗ctx, struct hostapd_radius_servers ∗conf)
- void **radius_client_deinit** (struct radius_client_data ∗radius)
- void **radius_client_flush_auth** (struct radius_client_data ∗radius, u8 ∗addr)

- int **radius_client_get_mib** (struct radius_client_data ∗radius, char ∗buf, size_t buflen)
- radius_client_data ∗ **radius_client_reconfig** (struct radius_client_data ∗old, void ∗ctx, struct hostapd_radius_servers ∗oldconf, struct hostapd_radius_servers ∗newconf)

6.120.1 Detailed Description

hostapd / RADIUS client

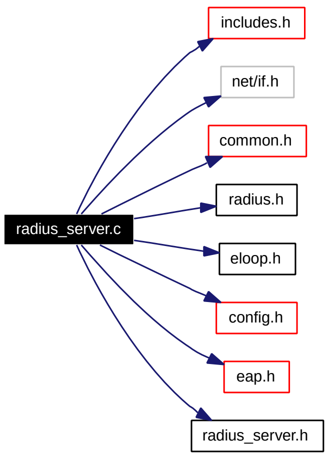
Definition in file radius_client.h.

6.121 radius_server.c File Reference

hostapd / RADIUS authentication server

```
#include "includes.h"
#include <net/if.h>
#include "common.h"
#include "radius.h"
#include "eloop.h"
#include "config.h"
#include "eap.h"
#include "radius_server.h"
```

Include dependency graph for radius_server.c:



Defines

- #define **RADIUS_SESSION_TIMEOUT** 60
- #define **RADIUS_MAX_SESSION** 100
- #define **RADIUS_MAX_MSG_LEN** 3000
- #define **RADIUS_DEBUG**(args...) wpa_printf(MSG_DEBUG, "RADIUS SRV: " args)
- #define **RADIUS_ERROR**(args...) wpa_printf(MSG_ERROR, "RADIUS SRV: " args)
- #define **RADIUS_DUMP**(args...) wpa_hexdump(MSG_MSGDUMP, "RADIUS SRV: " args)
- #define **RADIUS_DUMP_ASCII**(args...) wpa_hexdump_ascii(MSG_MSGDUMP, "RADIUS SRV: " args)

Functions

- radius_server_data ∗ **radius_server_init** (struct radius_server_conf ∗conf)
- void **radius_server_deinit** (struct radius_server_data ∗data)
- int **radius_server_get_mib** (struct radius_server_data ∗data, char ∗buf, size_t buflen)
- void **radius_server_eap_pending_cb** (struct radius_server_data ∗data, void ∗ctx)

Variables

- int **wpa_debug_level**

6.121.1 Detailed Description

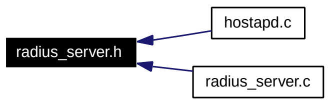
hostapd / RADIUS authentication server

Definition in file radius_server.c.

6.122 radius_server.h File Reference

hostapd / RADIUS authentication server

This graph shows which files directly or indirectly include this file:



Functions

- radius_server_data ∗ **radius_server_init** (struct radius_server_conf ∗conf)
- void **radius_server_deinit** (struct radius_server_data ∗data)
- int **radius_server_get_mib** (struct radius_server_data ∗data, char ∗buf, size_t buflen)
- void **radius_server_eap_pending_cb** (struct radius_server_data ∗data, void ∗ctx)

6.122.1 Detailed Description

hostapd / RADIUS authentication server

Copyright

Copyright (c) 2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

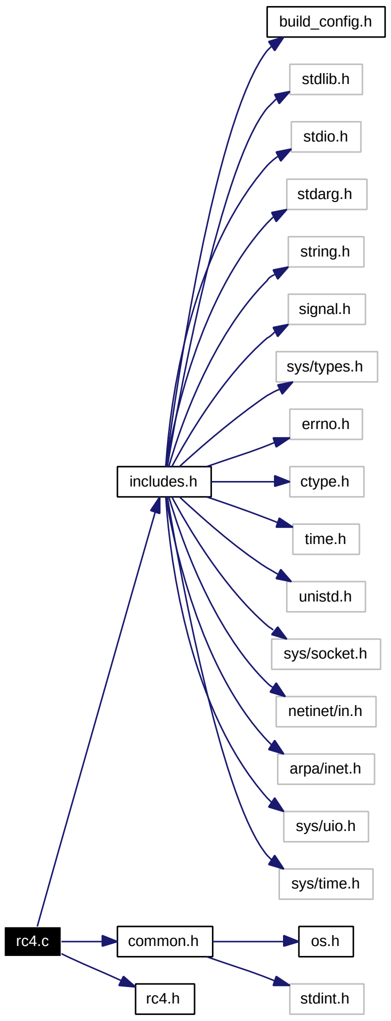
Definition in file radius_server.h.

6.123 rc4.c File Reference

RC4 stream cipher.

```
#include "includes.h"
#include "common.h"
#include "rc4.h"
```

Include dependency graph for rc4.c:



Defines

- #define **S_SWAP**(a, b) do { u8 t = S[a]; S[a] = S[b]; S[b] = t; } while(0)

Functions

- void rc4_skip (const u8 ∗key, size_t keylen, size_t skip, u8 ∗data, size_t data_len)

  XOR RC4 stream to given data with skip-stream-start.

- void rc4 (u8 ∗buf, size_t len, const u8 ∗key, size_t key_len)

  XOR RC4 stream to given data.

6.123.1 Detailed Description

RC4 stream cipher.

Copyright

Copyright (c) 2002-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file rc4.c.

6.123.2 Function Documentation

6.123.2.1 void rc4 (u8 ∗ *buf*, size_t *len*, const u8 ∗ *key*, size_t *key_len*)

XOR RC4 stream to given data.

Parameters:

buf data to be XOR'ed with RC4 stream

len buf length

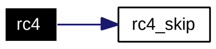
key RC4 key

key_len RC4 key length

Generate RC4 pseudo random stream for the given key and XOR this with the data buffer to perform RC4 encryption/decryption.

Definition at line 86 of file rc4.c.

Here is the call graph for this function:



6.123.2.2 void rc4_skip (const u8 ∗ *key*, size_t *keylen*, size_t *skip*, u8 ∗ *data*, size_t *data_len*)

XOR RC4 stream to given data with skip-stream-start.

Parameters:

key RC4 key

keylen RC4 key length

skip number of bytes to skip from the beginning of the RC4 stream

data data to be XOR'ed with RC4 stream

data_len buf length

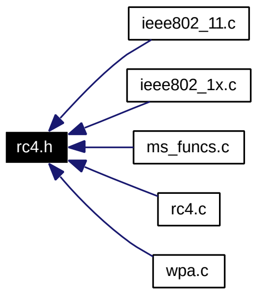
Generate RC4 pseudo random stream for the given key, skip beginning of the stream, and XOR the end result with the data buffer to perform RC4 encryption/decryption.

Definition at line 36 of file rc4.c.

6.124 rc4.h File Reference

RC4 stream cipher.

This graph shows which files directly or indirectly include this file:



Functions

- void rc4_skip (const u8 ∗key, size_t keylen, size_t skip, u8 ∗data, size_t data_len)

  XOR RC4 stream to given data with skip-stream-start.

- void rc4 (u8 ∗buf, size_t len, const u8 ∗key, size_t key_len)

  XOR RC4 stream to given data.

6.124.1 Detailed Description

RC4 stream cipher.

Copyright

Copyright (c) 2002-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file rc4.h.

6.124.2 Function Documentation

6.124.2.1 void rc4 (u8 ∗ *buf*, size_t *len*, const u8 ∗ *key*, size_t *key_len*)

XOR RC4 stream to given data.

Parameters:

buf data to be XOR'ed with RC4 stream

len buf length

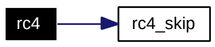
key RC4 key

key_len RC4 key length

Generate RC4 pseudo random stream for the given key and XOR this with the data buffer to perform RC4 encryption/decryption.

Definition at line 86 of file rc4.c.

Here is the call graph for this function:



6.124.2.2 void rc4_skip (const u8 ∗ *key*, size_t *keylen*, size_t *skip*, u8 ∗ *data*, size_t *data_len*)

XOR RC4 stream to given data with skip-stream-start.

Parameters:

key RC4 key

keylen RC4 key length

skip number of bytes to skip from the beginning of the RC4 stream

data data to be XOR'ed with RC4 stream

data_len buf length

Generate RC4 pseudo random stream for the given key, skip beginning of the stream, and XOR the end result with the data buffer to perform RC4 encryption/decryption.

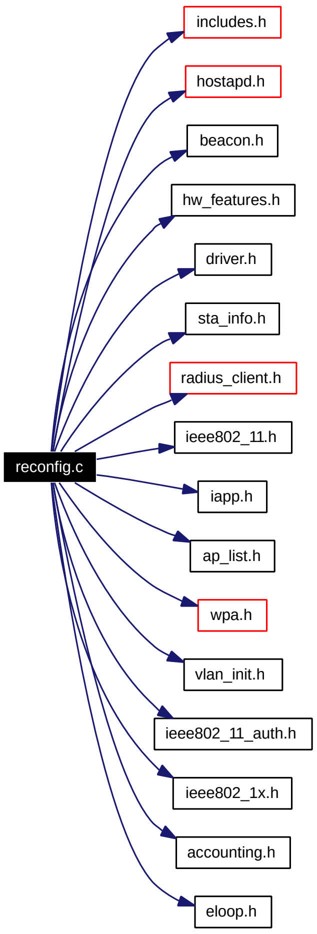
Definition at line 36 of file rc4.c.

6.125 reconfig.c File Reference

hostapd / Configuration reloading

```
#include "includes.h"
#include "hostapd.h"
#include "beacon.h"
#include "hw_features.h"
#include "driver.h"
#include "sta_info.h"
#include "radius_client.h"
#include "ieee802_11.h"
#include "iapp.h"
#include "ap_list.h"
#include "wpa.h"
#include "vlan_init.h"
#include "ieee802_11_auth.h"
#include "ieee802_1x.h"
#include "accounting.h"
#include "eloop.h"
```

Include dependency graph for reconfig.c:



Functions

- int hostapd_config_reload_start (struct hostapd_iface ∗hapd_iface, hostapd_iface_cb cb)

  Start reconfiguration of an interface.

6.125.1 Detailed Description

hostapd / Configuration reloading

Copyright

Copyright 2002-2003, Jouni Malinen <jkmaline@cc.hut.fi> Copyright 2002-2004, Instant802 Networks, Inc. Copyright 2005-2006, Devicescape Software, Inc. All Rights Reserved.

Definition in file reconfig.c.

6.125.2 Function Documentation

6.125.2.1 int hostapd_config_reload_start (struct hostapd_iface ∗ *hapd_iface*, hostapd_iface_cb *cb*)

Start reconfiguration of an interface.

Parameters:

hapd_iface Pointer to hostapd interface data

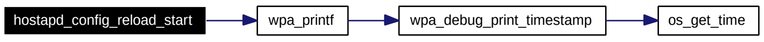
cb Function to be called back when done. The status indicates: 0 = success, new configuration in use; -1 = failed to update configuraiton, old configuration in use; -2 = failed to update configuration and failed to recover; caller should cleanup and terminate hostapd

Returns:

0 = reconfiguration started; -1 = failed to update configuration, old configuration in use; -2 = failed to update configuration and failed to recover; caller should cleanup and terminate hostapd

Definition at line 628 of file reconfig.c.

Here is the call graph for this function:

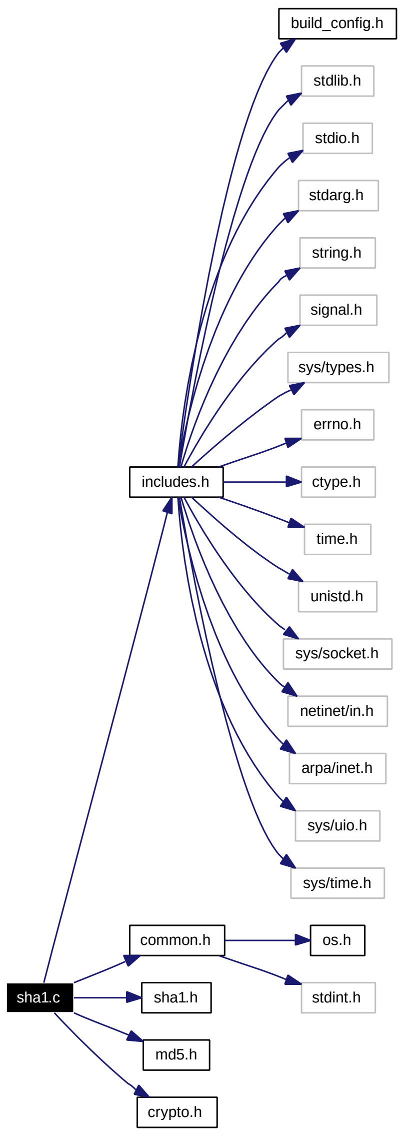


6.126 sha1.c File Reference

SHA1 hash implementation and interface functions.

```
#include "includes.h"
#include "common.h"
#include "sha1.h"
#include "md5.h"
#include "crypto.h"
```

Include dependency graph for sha1.c:



Functions

- void hmac_sha1_vector (const u8 ∗key, size_t key_len, size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

  HMAC-SHA1 over data vector (RFC 2104).

- void hmac_sha1 (const u8 ∗key, size_t key_len, const u8 ∗data, size_t data_len, u8 ∗mac)

  HMAC-SHA1 over data buffer (RFC 2104).

- void sha1_prf (const u8 ∗key, size_t key_len, const char ∗label, const u8 ∗data, size_t data_len, u8 ∗buf, size_t buf_len)

  SHA1-based Pseudo-Random Function (PRF) (IEEE 802.11i, 8.5.1.1).

- void sha1_t_prf (const u8 ∗key, size_t key_len, const char ∗label, const u8 ∗seed, size_t seed_len, u8 ∗buf, size_t buf_len)

  EAP-FAST Pseudo-Random Function (T-PRF).

- int tls_prf (const u8 ∗secret, size_t secret_len, const char ∗label, const u8 ∗seed, size_t seed_len, u8 ∗out, size_t outlen)

  Pseudo-Random Function for TLS (TLS-PRF, RFC 2246).

- void pbkdf2_sha1 (const char ∗passphrase, const char ∗ssid, size_t ssid_len, int iterations, u8 ∗buf, size_t buflen)

  SHA1-based key derivation function (PBKDF2) for IEEE 802.11i.

6.126.1 Detailed Description

SHA1 hash implementation and interface functions.

Copyright

Definition in file sha1.c.

6.126.2 Function Documentation

6.126.2.1 void hmac_sha1 (const u8 ∗ *key*, size_t *key_len*, const u8 ∗ *data*, size_t *data_len*, u8 ∗ *mac*)

HMAC-SHA1 over data buffer (RFC 2104).

Parameters:

key Key for HMAC operations

key_len Length of the key in bytes

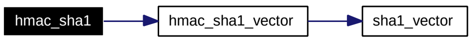
data Pointers to the data area

data_len Length of the data area

mac Buffer for the hash (20 bytes)

Definition at line 106 of file sha1.c.

Here is the call graph for this function:



6.126.2.2 void hmac_sha1_vector (const u8 ∗ *key*, size_t *key_len*, size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

HMAC-SHA1 over data vector (RFC 2104).

Parameters:

key Key for HMAC operations

key_len Length of the key in bytes

num_elem Number of elements in the data vector

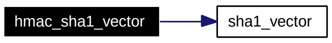
addr Pointers to the data areas

len Lengths of the data blocks

mac Buffer for the hash (20 bytes)

Definition at line 34 of file sha1.c.

Here is the call graph for this function:



6.126.2.3 void pbkdf2_sha1 (const char ∗ *passphrase*, const char ∗ *ssid*, size_t *ssid_len*, int *iterations*, u8 ∗ *buf*, size_t *buflen*)

SHA1-based key derivation function (PBKDF2) for IEEE 802.11i.

Parameters:

passphrase ASCII passphrase

ssid SSID

ssid_len SSID length in bytes

interations Number of iterations to run

buf Buffer for the generated key

buflen Length of the buffer in bytes

This function is used to derive PSK for WPA-PSK. For this protocol, iterations is set to 4096 and buflen to 32. This function is described in IEEE Std 802.11-2004, Clause H.4. The main construction is from PKCS#5 v2.0.

Definition at line 356 of file sha1.c.

6.126.2.4 void sha1_prf (const u8 ∗ *key*, size_t *key_len*, const char ∗ *label*, const u8 ∗ *data*, size_t *data_len*, u8 ∗ *buf*, size_t *buf_len*)

SHA1-based Pseudo-Random Function (PRF) (IEEE 802.11i, 8.5.1.1).

Parameters:

key Key for PRF

key_len Length of the key in bytes

label A unique label for each purpose of the PRF

data Extra data to bind into the key

data_len Length of the data

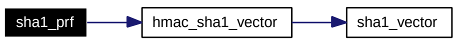
buf Buffer for the generated pseudo-random key

buf_len Number of bytes of key to generate

This function is used to derive new, cryptographically separate keys from a given key (e.g., PMK in IEEE 802.11i).

Definition at line 127 of file sha1.c.

Here is the call graph for this function:



6.126.2.5 void sha1_t_prf (const u8 ∗ *key*, size_t *key_len*, const char ∗ *label*, const u8 ∗ *seed*, size_t *seed_len*, u8 ∗ *buf*, size_t *buf_len*)

EAP-FAST Pseudo-Random Function (T-PRF).

Parameters:

key Key for PRF

key_len Length of the key in bytes

label A unique label for each purpose of the PRF

seed Seed value to bind into the key

seed_len Length of the seed

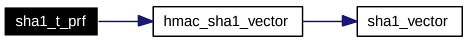
buf Buffer for the generated pseudo-random key

buf_len Number of bytes of key to generate

This function is used to derive new, cryptographically separate keys from a given key for EAP-FAST. T-PRF is defined in draft-cam-winget-eap-fast-02.txt, Appendix B.

Definition at line 179 of file sha1.c.

Here is the call graph for this function:



6.126.2.6 int tls_prf (const u8 ∗ *secret*, size_t *secret_len*, const char ∗ *label*, const u8 ∗ *seed*, size_t *seed_len*, u8 ∗ *out*, size_t *outlen*)

Pseudo-Random Function for TLS (TLS-PRF, RFC 2246).

Parameters:

secret Key for PRF

secret_len Length of the key in bytes

label A unique label for each purpose of the PRF

seed Seed value to bind into the key

seed_len Length of the seed

out Buffer for the generated pseudo-random key

outlen Number of bytes of key to generate

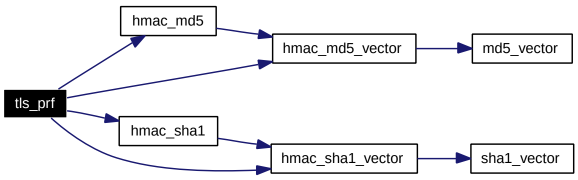
Returns:

0 on success, -1 on failure.

This function is used to derive new, cryptographically separate keys from a given key in TLS. This PRF is defined in RFC 2246, Chapter 5.

Definition at line 235 of file sha1.c.

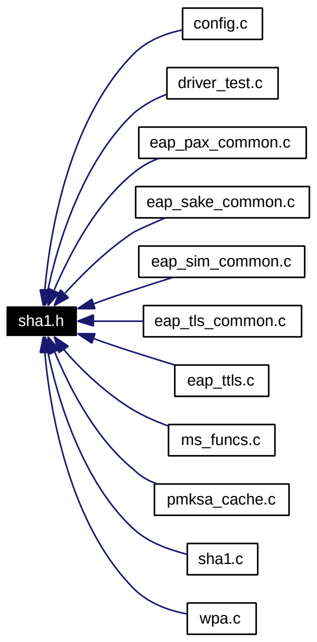
Here is the call graph for this function:



6.127 sha1.h File Reference

SHA1 hash implementation and interface functions.

This graph shows which files directly or indirectly include this file:



Defines

- #define **SHA1_MAC_LEN** 20

Functions

- void hmac_sha1_vector (const u8 ∗key, size_t key_len, size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

    HMAC-SHA1 over data vector (RFC 2104).

- void hmac_sha1 (const u8 ∗key, size_t key_len, const u8 ∗data, size_t data_len, u8 ∗mac)

    HMAC-SHA1 over data buffer (RFC 2104).

- void sha1_prf (const u8 ∗key, size_t key_len, const char ∗label, const u8 ∗data, size_t data_len, u8 ∗buf, size_t buf_len)

    SHA1-based Pseudo-Random Function (PRF) (IEEE 802.11i, 8.5.1.1).

- void sha1_t_prf (const u8 ∗key, size_t key_len, const char ∗label, const u8 ∗seed, size_t seed_len, u8 ∗buf, size_t buf_len)

    EAP-FAST Pseudo-Random Function (T-PRF).

- int tls_prf (const u8 ∗secret, size_t secret_len, const char ∗label, const u8 ∗seed, size_t seed_len, u8 ∗out, size_t outlen)

Pseudo-Random Function for TLS (TLS-PRF, RFC 2246).

- void pbkdf2_sha1 (const char ∗passphrase, const char ∗ssid, size_t ssid_len, int iterations, u8 ∗buf, size_t buflen)

  SHA1-based key derivation function (PBKDF2) for IEEE 802.11i.

6.127.1 Detailed Description

SHA1 hash implementation and interface functions.

Copyright

Copyright (c) 2003-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file sha1.h.

6.127.2 Function Documentation

6.127.2.1 void hmac_sha1 (const u8 ∗ *key*, size_t *key_len*, const u8 ∗ *data*, size_t *data_len*, u8 ∗ *mac*)

HMAC-SHA1 over data buffer (RFC 2104).

Parameters:

key Key for HMAC operations

key_len Length of the key in bytes

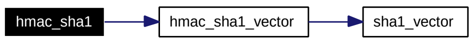
data Pointers to the data area

data_len Length of the data area

mac Buffer for the hash (20 bytes)

Definition at line 106 of file sha1.c.

Here is the call graph for this function:



6.127.2.2 void hmac_sha1_vector (const u8 ∗ *key*, size_t *key_len*, size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

HMAC-SHA1 over data vector (RFC 2104).

Parameters:

key Key for HMAC operations

key_len Length of the key in bytes

num_elem Number of elements in the data vector

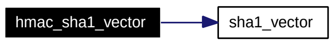
addr Pointers to the data areas

len Lengths of the data blocks

mac Buffer for the hash (20 bytes)

Definition at line 34 of file sha1.c.

Here is the call graph for this function:



6.127.2.3 void pbkdf2_sha1 (const char ∗ *passphrase*, const char ∗ *ssid*, size_t *ssid_len*, int *iterations*, u8 ∗ *buf*, size_t *buflen*)

SHA1-based key derivation function (PBKDF2) for IEEE 802.11i.

Parameters:

passphrase ASCII passphrase

ssid SSID

ssid_len SSID length in bytes

interations Number of iterations to run

buf Buffer for the generated key

buflen Length of the buffer in bytes

This function is used to derive PSK for WPA-PSK. For this protocol, iterations is set to 4096 and buflen to 32. This function is described in IEEE Std 802.11-2004, Clause H.4. The main construction is from PKCS#5 v2.0.

Definition at line 356 of file sha1.c.

6.127.2.4 void sha1_prf (const u8 ∗ *key*, size_t *key_len*, const char ∗ *label*, const u8 ∗ *data*, size_t *data_len*, u8 ∗ *buf*, size_t *buf_len*)

SHA1-based Pseudo-Random Function (PRF) (IEEE 802.11i, 8.5.1.1).

Parameters:

key Key for PRF

key_len Length of the key in bytes

label A unique label for each purpose of the PRF

data Extra data to bind into the key

data_len Length of the data

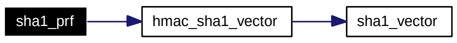
buf Buffer for the generated pseudo-random key

buf_len Number of bytes of key to generate

This function is used to derive new, cryptographically separate keys from a given key (e.g., PMK in IEEE 802.11i).

Definition at line 127 of file sha1.c.

Here is the call graph for this function:



6.127.2.5 void sha1_t_prf (const u8 ∗ *key*, size_t *key_len*, const char ∗ *label*, const u8 ∗ *seed*, size_t *seed_len*, u8 ∗ *buf*, size_t *buf_len*)

EAP-FAST Pseudo-Random Function (T-PRF).

Parameters:

key Key for PRF

key_len Length of the key in bytes

label A unique label for each purpose of the PRF

seed Seed value to bind into the key

seed_len Length of the seed

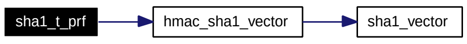
buf Buffer for the generated pseudo-random key

buf_len Number of bytes of key to generate

This function is used to derive new, cryptographically separate keys from a given key for EAP-FAST. T-PRF is defined in draft-cam-winget-eap-fast-02.txt, Appendix B.

Definition at line 179 of file sha1.c.

Here is the call graph for this function:



6.127.2.6 int tls_prf (const u8 ∗ *secret*, size_t *secret_len*, const char ∗ *label*, const u8 ∗ *seed*, size_t *seed_len*, u8 ∗ *out*, size_t *outlen*)

Pseudo-Random Function for TLS (TLS-PRF, RFC 2246).

Parameters:

secret Key for PRF

secret_len Length of the key in bytes

label A unique label for each purpose of the PRF

seed Seed value to bind into the key

seed_len Length of the seed

out Buffer for the generated pseudo-random key

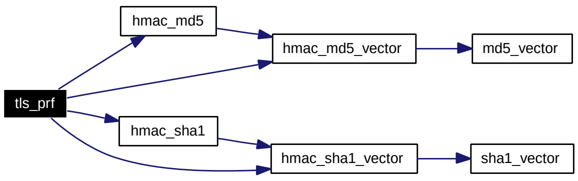
outlen Number of bytes of key to generate

Returns:
0 on success, -1 on failure.

This function is used to derive new, cryptographically separate keys from a given key in TLS. This PRF is defined in RFC 2246, Chapter 5.

Definition at line 235 of file sha1.c.

Here is the call graph for this function:



6.128 sha256.c File Reference

SHA-256 hash implementation and interface functions.

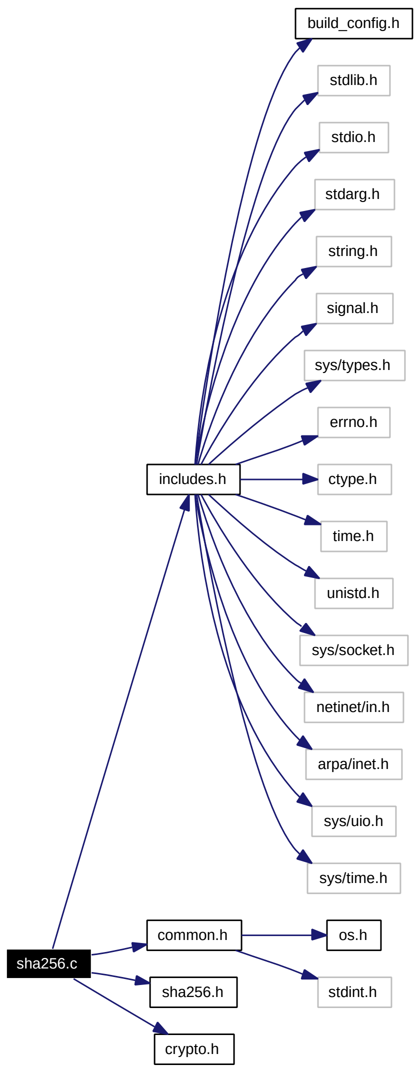
`#include "includes.h"`

`#include "common.h"`

`#include "sha256.h"`

`#include "crypto.h"`

Include dependency graph for sha256.c:



Functions

- void hmac_sha256_vector (const u8 ∗key, size_t key_len, size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

  HMAC-SHA256 over data vector (RFC 2104).

- void hmac_sha256 (const u8 ∗key, size_t key_len, const u8 ∗data, size_t data_len, u8 ∗mac)

  HMAC-SHA256 over data buffer (RFC 2104).

- void sha256_prf (const u8 ∗key, size_t key_len, const char ∗label, const u8 ∗data, size_t data_len, u8 ∗buf, size_t buf_len)

  SHA256-based Pseudo-Random Function (IEEE 802.11r, 8.5A.3).

6.128.1 Detailed Description

SHA-256 hash implementation and interface functions.

Copyright

Copyright (c) 2003-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

Definition in file sha256.c.

6.128.2 Function Documentation

6.128.2.1 void hmac_sha256 (const u8 ∗ *key*, size_t *key_len*, const u8 ∗ *data*, size_t *data_len*, u8 ∗ *mac*)

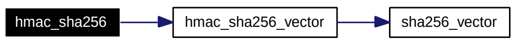
HMAC-SHA256 over data buffer (RFC 2104).

Parameters:

key	Key for HMAC operations
key_len	Length of the key in bytes
data	Pointers to the data area
data_len	Length of the data area
mac	Buffer for the hash (20 bytes)

Definition at line 105 of file sha256.c.

Here is the call graph for this function:



6.128.2.2 void hmac_sha256_vector (const u8 ∗ *key*, size_t *key_len*, size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

HMAC-SHA256 over data vector (RFC 2104).

Parameters:

key Key for HMAC operations

key_len Length of the key in bytes

num_elem Number of elements in the data vector

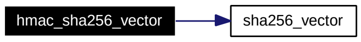
addr Pointers to the data areas

len Lengths of the data blocks

mac Buffer for the hash (32 bytes)

Definition at line 33 of file sha256.c.

Here is the call graph for this function:



6.128.2.3 void sha256_prf (const u8 ∗ *key*, size_t *key_len*, const char ∗ *label*, const u8 ∗ *data*, size_t *data_len*, u8 ∗ *buf*, size_t *buf_len*)

SHA256-based Pseudo-Random Function (IEEE 802.11r, 8.5A.3).

Parameters:

key Key for PRF

key_len Length of the key in bytes

label A unique label for each purpose of the PRF

data Extra data to bind into the key

data_len Length of the data

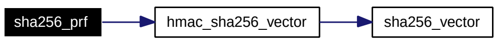
buf Buffer for the generated pseudo-random key

buf_len Number of bytes of key to generate

This function is used to derive new, cryptographically separate keys from a given key.

Definition at line 126 of file sha256.c.

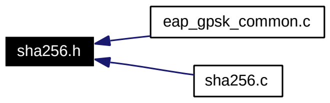
Here is the call graph for this function:



6.129 sha256.h File Reference

SHA256 hash implementation and interface functions.

This graph shows which files directly or indirectly include this file:



Defines

- #define **SHA256_MAC_LEN** 32

Functions

- void hmac_sha256_vector (const u8 ∗key, size_t key_len, size_t num_elem, const u8 ∗addr[ ], const size_t ∗len, u8 ∗mac)

  HMAC-SHA256 over data vector (RFC 2104).

- void hmac_sha256 (const u8 ∗key, size_t key_len, const u8 ∗data, size_t data_len, u8 ∗mac)

  HMAC-SHA256 over data buffer (RFC 2104).

- void sha256_prf (const u8 ∗key, size_t key_len, const char ∗label, const u8 ∗data, size_t data_len, u8 ∗buf, size_t buf_len)

  SHA256-based Pseudo-Random Function (IEEE 802.11r, 8.5A.3).

6.129.1 Detailed Description

SHA256 hash implementation and interface functions.

Copyright

Copyright (c) 2003-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file sha256.h.

6.129.2 Function Documentation

6.129.2.1 void hmac_sha256 (const u8 ∗ *key*, size_t *key_len*, const u8 ∗ *data*, size_t *data_len*, u8 ∗ *mac*)

HMAC-SHA256 over data buffer (RFC 2104).

Parameters:

key Key for HMAC operations

key_len Length of the key in bytes

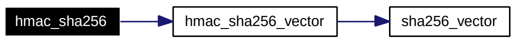
data Pointers to the data area

data_len Length of the data area

mac Buffer for the hash (20 bytes)

Definition at line 105 of file sha256.c.

Here is the call graph for this function:



6.129.2.2 void hmac_sha256_vector (const u8 ∗ *key*, size_t *key_len*, size_t *num_elem*, const u8 ∗ *addr*[ ], const size_t ∗ *len*, u8 ∗ *mac*)

HMAC-SHA256 over data vector (RFC 2104).

Parameters:

key Key for HMAC operations

key_len Length of the key in bytes

num_elem Number of elements in the data vector

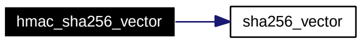
addr Pointers to the data areas

len Lengths of the data blocks

mac Buffer for the hash (32 bytes)

Definition at line 33 of file sha256.c.

Here is the call graph for this function:



6.129.2.3 void sha256_prf (const u8 ∗ *key*, size_t *key_len*, const char ∗ *label*, const u8 ∗ *data*, size_t *data_len*, u8 ∗ *buf*, size_t *buf_len*)

SHA256-based Pseudo-Random Function (IEEE 802.11r, 8.5A.3).

Parameters:

key Key for PRF

key_len Length of the key in bytes

label A unique label for each purpose of the PRF

data Extra data to bind into the key

data_len Length of the data

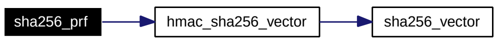
buf Buffer for the generated pseudo-random key

buf_len Number of bytes of key to generate

This function is used to derive new, cryptographically separate keys from a given key.

Definition at line 126 of file sha256.c.

Here is the call graph for this function:

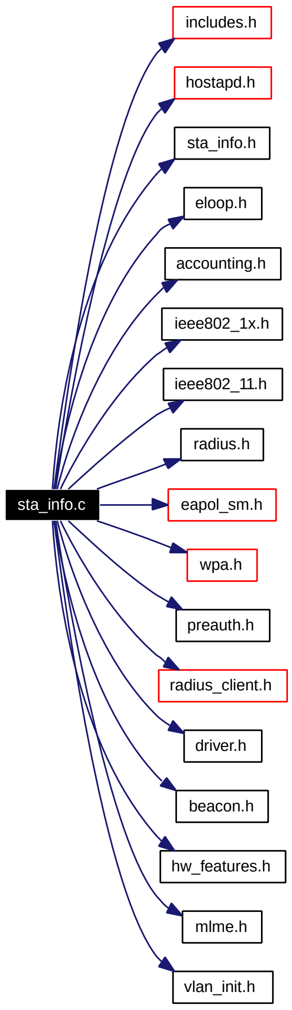


6.130 sta_info.c File Reference

hostapd / Station table

```
#include "includes.h"
#include "hostapd.h"
#include "sta_info.h"
#include "eloop.h"
#include "accounting.h"
#include "ieee802_1x.h"
#include "ieee802_11.h"
#include "radius.h"
#include "eapol_sm.h"
#include "wpa.h"
#include "preauth.h"
#include "radius_client.h"
#include "driver.h"
#include "beacon.h"
#include "hw_features.h"
#include "mlme.h"
#include "vlan_init.h"
```

Include dependency graph for sta_info.c:



Functions

- int **ap_for_each_sta** (struct hostapd_data ∗hapd, int(∗cb)(struct hostapd_data ∗hapd, struct sta_info ∗sta, void ∗ctx), void ∗ctx)
- sta_info ∗ **ap_get_sta** (struct hostapd_data ∗hapd, const u8 ∗sta)
- void **ap_sta_hash_add** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ap_free_sta** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **hostapd_free_stas** (struct hostapd_data ∗hapd)
- void **ap_handle_timer** (void ∗eloop_ctx, void ∗timeout_ctx)
- void **ap_sta_session_timeout** (struct hostapd_data ∗hapd, struct sta_info ∗sta, u32 session_timeout)
- void **ap_sta_no_session_timeout** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- sta_info ∗ **ap_sta_add** (struct hostapd_data ∗hapd, const u8 ∗addr)
- void **ap_sta_disassociate** (struct hostapd_data ∗hapd, struct sta_info ∗sta, u16 reason)
- void **ap_sta_deauthenticate** (struct hostapd_data ∗hapd, struct sta_info ∗sta, u16 reason)
- int **ap_sta_bind_vlan** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int old_vlanid)

6.130.1 Detailed Description

hostapd / Station table

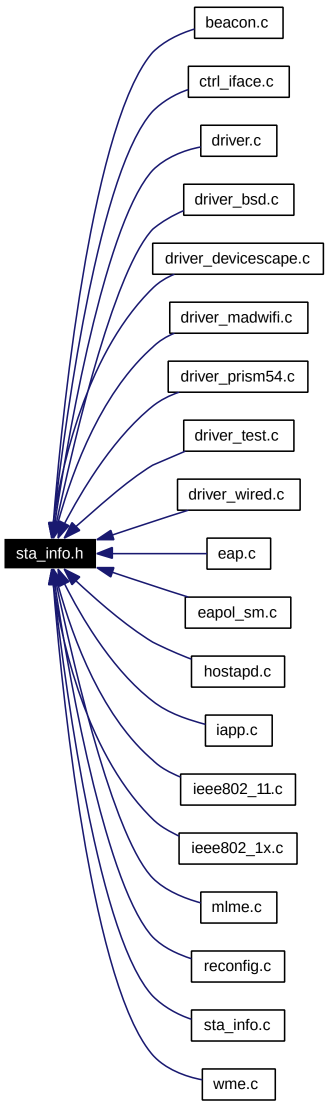
Copyright

Definition in file sta_info.c.

6.131 sta_info.h File Reference

hostapd / Station table

This graph shows which files directly or indirectly include this file:



Functions

- int **ap_for_each_sta** (struct hostapd_data ∗hapd, int(∗cb)(struct hostapd_data ∗hapd, struct sta_info ∗sta, void ∗ctx), void ∗ctx)
- sta_info ∗ **ap_get_sta** (struct hostapd_data ∗hapd, const u8 ∗sta)
- void **ap_sta_hash_add** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **ap_free_sta** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **hostapd_free_stas** (struct hostapd_data ∗hapd)
- void **ap_handle_timer** (void ∗eloop_ctx, void ∗timeout_ctx)

- void **ap_sta_session_timeout** (struct hostapd_data ∗hapd, struct sta_info ∗sta, u32 session_timeout)
- void **ap_sta_no_session_timeout** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- sta_info ∗ **ap_sta_add** (struct hostapd_data ∗hapd, const u8 ∗addr)
- void **ap_sta_disassociate** (struct hostapd_data ∗hapd, struct sta_info ∗sta, u16 reason)
- void **ap_sta_deauthenticate** (struct hostapd_data ∗hapd, struct sta_info ∗sta, u16 reason)
- int **ap_sta_bind_vlan** (struct hostapd_data ∗hapd, struct sta_info ∗sta, int old_vlanid)

6.131.1 Detailed Description

hostapd / Station table

Copyright

Copyright (c) 2002-2004, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

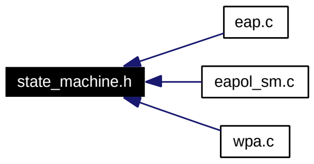
See README and COPYING for more details.

Definition in file sta_info.h.

6.132 state_machine.h File Reference

wpa_supplicant/hostapd - State machine definitions

This graph shows which files directly or indirectly include this file:



Defines

- #define SM_STATE(machine, state)

  Declaration of a state machine function.

- #define SM_ENTRY(machine, state)

  State machine function entry point.

- #define SM_ENTRY_M(machine, _state, data)

  State machine function entry point for state machine group.

- #define SM_ENTRY_MA(machine, _state, data)

  State machine function entry point for state machine group.

- #define SM_ENTER(machine, state) sm_ ## machine ## _ ## state ## _Enter(sm, 0)

  Enter a new state machine state.

- #define SM_ENTER_GLOBAL(machine, state) sm_ ## machine ## _ ## state ## _Enter(sm, 1)

  Enter a new state machine state based on global rule.

- #define SM_STEP(machine) static void sm_ ## machine ## _Step(STATE_MACHINE_DATA ∗sm)

  Declaration of a state machine step function.

- #define SM_STEP_RUN(machine) sm_ ## machine ## _Step(sm)

  Call the state machine step function.

6.132.1 Detailed Description

wpa_supplicant/hostapd - State machine definitions

Copyright

Copyright (c) 2002-2005, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

This file includes a set of pre-processor macros that can be used to implement a state machine. In addition to including this header file, each file implementing a state machine must define STATE_MACHINE_-DATA to be the data structure including state variables (enum <machine>_state, Boolean changed), and STATE_MACHINE_DEBUG_PREFIX to be a string that is used as a prefix for all debug messages. If SM_ENTRY_MA macro is used to define a group of state machines with shared data structure, STATE_-MACHINE_ADDR needs to be defined to point to the MAC address used in debug output. SM_ENTRY_M macro can be used to define similar group of state machines without this additional debug info.

Definition in file state_machine.h.

6.132.2 Define Documentation

6.132.2.1 #define SM_ENTER(machine, state) sm_ ## machine ## _ ## state ## _Enter(sm, 0)

Enter a new state machine state.

Parameters:

machine State machine name

state State machine state

This macro expands to a function call to a state machine function defined with SM_STATE macro. SM_-ENTER is used in a state machine step function to move the state machine to a new state.

Definition at line 113 of file state_machine.h.

6.132.2.2 #define SM_ENTER_GLOBAL(machine, state) sm_ ## machine ## _ ## state ## _Enter(sm, 1)

Enter a new state machine state based on global rule.

Parameters:

machine State machine name

state State machine state

This macro is like SM_ENTER, but this is used when entering a new state based on a global (not specific to any particular state) rule. A separate macro is used to avoid unwanted debug message floods when the same global rule is forcing a state machine to remain in on state.

Definition at line 127 of file state_machine.h.

6.132.2.3 #define SM_ENTRY(machine, state)

Value:

```
if (!global || sm->machine ## _state != machine ## _ ## state) { \
        sm->changed = TRUE; \
        wpa_printf(MSG_DEBUG, STATE_MACHINE_DEBUG_PREFIX ": " #machine \
                   " entering state " #state); \
} \
sm->machine ## _state = machine ## _ ## state;
```

State machine function entry point.

Parameters:

machine State machine name

state State machine state

This macro is used inside each state machine function declared with SM_STATE. SM_ENTRY should be in the beginning of the function body, but after declaration of possible local variables. This macro prints debug information about state transition and update the state machine state.

Definition at line 55 of file state_machine.h.

6.132.2.4 #define SM_ENTRY_M(machine, _state, data)

Value:

```
if (!global || sm->data ## _ ## state != machine ## _ ## _state) { \
        sm->changed = TRUE; \
        wpa_printf(MSG_DEBUG, STATE_MACHINE_DEBUG_PREFIX ": " \
                   #machine " entering state " #_state); \
} \
sm->data ## _ ## state = machine ## _ ## _state;
```

State machine function entry point for state machine group.

Parameters:

machine State machine name

_state State machine state

data State variable prefix (full variable: <prefix>_state)

This macro is like SM_ENTRY, but for state machine groups that use a shared data structure for more than one state machine. Both machine and prefix parameters are set to "sub-state machine" name. prefix is used to allow more than one state variable to be stored in the same data structure.

Definition at line 75 of file state_machine.h.

6.132.2.5 #define SM_ENTRY_MA(machine, _state, data)

Value:

```
if (!global || sm->data ## _ ## state != machine ## _ ## _state) { \
        sm->changed = TRUE; \
        wpa_printf(MSG_DEBUG, STATE_MACHINE_DEBUG_PREFIX ": " MACSTR " " \
                   #machine " entering state " #_state, \
                   MAC2STR(STATE_MACHINE_ADDR)); \
} \
sm->data ## _ ## state = machine ## _ ## _state;
```

State machine function entry point for state machine group.

Parameters:

machine State machine name

_state State machine state

data State variable prefix (full variable: <prefix>_state)

This macro is like SM_ENTRY_M, but a MAC address is included in debug output. STATE_MACHINE_-ADDR has to be defined to point to the MAC address to be included in debug.

Definition at line 94 of file state_machine.h.

6.132.2.6 #define SM_STATE(machine, state)

Value:

```
static void sm_ ## machine ## _ ## state ## _Enter(STATE_MACHINE_DATA *sm, \
        int global)
```

Declaration of a state machine function.

Parameters:

machine State machine name

state State machine state

This macro is used to declare a state machine function. It is used in place of a C function definition to declare functions to be run when the state is entered by calling SM_ENTER or SM_ENTER_GLOBAL.

Definition at line 40 of file state_machine.h.

6.132.2.7 #define SM_STEP(machine) static void sm_ ## machine ## _Step(STATE_MACHINE_DATA *sm)

Declaration of a state machine step function.

Parameters:

machine State machine name

This macro is used to declare a state machine step function. It is used in place of a C function definition to declare a function that is used to move state machine to a new state based on state variables. This function uses SM_ENTER and SM_ENTER_GLOBAL macros to enter new state.

Definition at line 140 of file state_machine.h.

6.132.2.8 #define SM_STEP_RUN(machine) sm_ ## machine ## _Step(sm)

Call the state machine step function.

Parameters:

machine State machine name

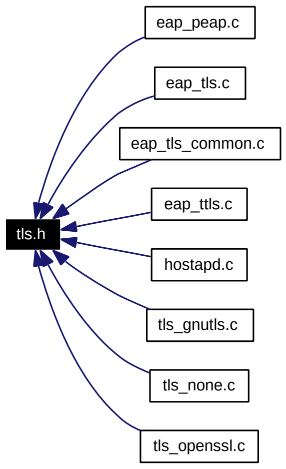
This macro expands to a function call to a state machine step function defined with SM_STEP macro.

Definition at line 151 of file state_machine.h.

6.133 tls.h File Reference

WPA Supplicant / SSL/TLS interface definition.

This graph shows which files directly or indirectly include this file:



Defines

- #define **TLS_CAPABILITY_IA** 0x0001

Enumerations

- enum { **TLS_SET_PARAMS_ENGINE_PRV_VERIFY_FAILED** = -3, **TLS_SET_PARAMS_-ENGINE_PRV_INIT_FAILED** = -2 }
- enum {

  TLS_CIPHER_NONE, **TLS_CIPHER_RC4_SHA**, **TLS_CIPHER_AES128_SHA**, **TLS_-CIPHER_RSA_DHE_AES128_SHA**,

  TLS_CIPHER_ANON_DH_AES128_SHA }

Functions

- void ∗ tls_init (const struct tls_config ∗conf)

  Initialize TLS library.

- void tls_deinit (void ∗tls_ctx)

  Deinitialize TLS library.

- int tls_get_errors (void ∗tls_ctx)

  Process pending errors.

- tls_connection ∗ tls_connection_init (void ∗tls_ctx)

  Initialize a new TLS connection.

- void tls_connection_deinit (void ∗tls_ctx, struct tls_connection ∗conn)

  Free TLS connection data.

- int tls_connection_established (void ∗tls_ctx, struct tls_connection ∗conn)

  Has the TLS connection been completed?

- int tls_connection_shutdown (void ∗tls_ctx, struct tls_connection ∗conn)

  Shutdown TLS connection.

- int tls_connection_set_params (void ∗tls_ctx, struct tls_connection ∗conn, const struct tls_-connection_params ∗params)

  Set TLS connection parameters.

- int tls_global_set_params (void ∗tls_ctx, const struct tls_connection_params ∗params)

  Set TLS parameters for all TLS connection.

- int tls_global_set_verify (void ∗tls_ctx, int check_crl)

  Set global certificate verification options.

- int tls_connection_set_verify (void ∗tls_ctx, struct tls_connection ∗conn, int verify_peer)

  Set certificate verification options.

- int tls_connection_set_ia (void ∗tls_ctx, struct tls_connection ∗conn, int tls_ia)

  Set TLS/IA parameters.

- int tls_connection_get_keys (void ∗tls_ctx, struct tls_connection ∗conn, struct tls_keys ∗keys)

  Get master key and random data from TLS connection.

- int tls_connection_prf (void ∗tls_ctx, struct tls_connection ∗conn, const char ∗label, int server_-random_first, u8 ∗out, size_t out_len)

  Use TLS-PRF to derive keying material.

- u8 ∗ tls_connection_handshake (void ∗tls_ctx, struct tls_connection ∗conn, const u8 ∗in_data, size_t in_len, size_t ∗out_len, u8 ∗∗appl_data, size_t ∗appl_data_len)

  Process TLS handshake (client side).

- u8 ∗ tls_connection_server_handshake (void ∗tls_ctx, struct tls_connection ∗conn, const u8 ∗in_-data, size_t in_len, size_t ∗out_len)

  Process TLS handshake (server side).

- int tls_connection_encrypt (void ∗tls_ctx, struct tls_connection ∗conn, const u8 ∗in_data, size_t in_-len, u8 ∗out_data, size_t out_len)

  Encrypt data into TLS tunnel.

- int tls_connection_decrypt (void ∗tls_ctx, struct tls_connection ∗conn, const u8 ∗in_data, size_t in_-len, u8 ∗out_data, size_t out_len)

  Decrypt data from TLS tunnel.

- int tls_connection_resumed (void ∗tls_ctx, struct tls_connection ∗conn)

  Was session resumption used.

- int tls_connection_set_master_key (void ∗tls_ctx, struct tls_connection ∗conn, const u8 ∗key, size_t key_len)

  Configure master secret for TLS connection.

- int tls_connection_set_cipher_list (void ∗tls_ctx, struct tls_connection ∗conn, u8 ∗ciphers)

  Configure acceptable cipher suites.

- int tls_get_cipher (void ∗tls_ctx, struct tls_connection ∗conn, char ∗buf, size_t buflen)

  Get current cipher name.

- int tls_connection_enable_workaround (void ∗tls_ctx, struct tls_connection ∗conn)

  Enable TLS workaround options.

- int tls_connection_client_hello_ext (void ∗tls_ctx, struct tls_connection ∗conn, int ext_type, const u8 ∗data, size_t data_len)

  Set TLS extension for ClientHello.

- int tls_connection_get_failed (void ∗tls_ctx, struct tls_connection ∗conn)

  Get connection failure status.

- int tls_connection_get_read_alerts (void ∗tls_ctx, struct tls_connection ∗conn)

  Get connection read alert status.

- int tls_connection_get_write_alerts (void ∗tls_ctx, struct tls_connection ∗conn)

  Get connection write alert status.

- int tls_connection_get_keyblock_size (void ∗tls_ctx, struct tls_connection ∗conn)

  Get TLS key_block size.

- unsigned int tls_capabilities (void ∗tls_ctx)

  Get supported TLS capabilities.

- int tls_connection_ia_send_phase_finished (void ∗tls_ctx, struct tls_connection ∗conn, int final, u8 ∗out_data, size_t out_len)

  Send a TLS/IA PhaseFinished message.

- int tls_connection_ia_final_phase_finished (void ∗tls_ctx, struct tls_connection ∗conn)

  Has final phase been completed.

- int tls_connection_ia_permute_inner_secret (void ∗tls_ctx, struct tls_connection ∗conn, const u8 ∗key, size_t key_len)

  Permute TLS/IA inner secret.

6.133.1 Detailed Description

WPA Supplicant / SSL/TLS interface definition.

Definition in file tls.h.

6.133.2 Function Documentation

6.133.2.1 unsigned int tls_capabilities (void ∗ *tls_ctx*)

Get supported TLS capabilities.

Parameters:

tls_ctx TLS context data from tls_init()

Returns:

Bit field of supported TLS capabilities (TLS_CAPABILITY_∗)

Definition at line 1231 of file tls_gnutls.c.

6.133.2.2 int tls_connection_client_hello_ext (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *ext_type*, const u8 ∗ *data*, size_t *data_len*)

Set TLS extension for ClientHello.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

ext_type Extension type

data Extension payload (NULL to remove extension)

data_len Extension payload length

Returns:

0 on success, -1 on failure

Definition at line 1190 of file tls_gnutls.c.

6.133.2.3 int tls_connection_decrypt (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, u8 ∗ *out_data*, size_t *out_len*)

Decrypt data from TLS tunnel.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Pointer to input buffer (encrypted TLS data)

in_len Input buffer length

out_data Pointer to output buffer (decrypted data from TLS tunnel)

out_len Maximum out_data length

Returns:

Number of bytes written to out_data, -1 on failure

This function is used after TLS handshake has been completed successfully to receive data from the encrypted tunnel.

Definition at line 1067 of file tls_gnutls.c.

6.133.2.4 void tls_connection_deinit (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Free TLS connection data.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Release all resources allocated for TLS connection.

Definition at line 361 of file tls_gnutls.c.

6.133.2.5 int tls_connection_enable_workaround (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Enable TLS workaround options.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

0 on success, -1 on failure

This function is used to enable connection-specific workaround options for buffer SSL/TLS implementations.

Definition at line 1182 of file tls_gnutls.c.

6.133.2.6 int tls_connection_encrypt (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, u8 ∗ *out_data*, size_t *out_len*)

Encrypt data into TLS tunnel.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Pointer to plaintext data to be encrypted

in_len Input buffer length

out_data Pointer to output buffer (encrypted TLS data)

out_len Maximum out_data length

Returns:

Number of bytes written to out_data, -1 on failure

This function is used after TLS handshake has been completed successfully to send data in the encrypted tunnel.

Definition at line 1038 of file tls_gnutls.c.

6.133.2.7 int tls_connection_established (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Has the TLS connection been completed?

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

1 if TLS connection has been completed, 0 if not.

Definition at line 388 of file tls_gnutls.c.

6.133.2.8 int tls_connection_get_failed (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get connection failure status.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns >0 if connection has failed, 0 if not.

Definition at line 1199 of file tls_gnutls.c.

6.133.2.9 int tls_connection_get_keyblock_size (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get TLS key_block size.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

Size of the key_block for the negotiated cipher suite or -1 on failure

Definition at line 1223 of file tls_gnutls.c.

6.133.2.10 int tls_connection_get_keys (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, struct tls_keys ∗ *keys*)

Get master key and random data from TLS connection.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

keys Structure of key/random data (filled on success)

Returns:

0 on success, -1 on failure

Definition at line 790 of file tls_gnutls.c.

6.133.2.11 int tls_connection_get_read_alerts (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get connection read alert status.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

Number of times a fatal read (remote end reported error) has happened during this connection.

Definition at line 1207 of file tls_gnutls.c.

6.133.2.12 int tls_connection_get_write_alerts (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get connection write alert status.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

Number of times a fatal write (locally detected error) has happened during this connection.

Definition at line 1215 of file tls_gnutls.c.

6.133.2.13 u8∗ tls_connection_handshake (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, size_t ∗ *out_len*, u8 ∗∗ *appl_data*, size_t ∗ *appl_data_len*)

Process TLS handshake (client side).

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Input data from TLS peer

in_len Input data length

out_len Length of the output buffer.

appl_data Pointer to application data pointer, or NULL if dropped

appl_data_len Pointer to variable that is set to appl_data length

Returns:

Pointer to output data, NULL on failure

Caller is responsible for freeing returned output data. If the final handshake message includes application data, this is decrypted and appl_data (if not NULL) is set to point this data. Caller is responsible for freeing appl_data.

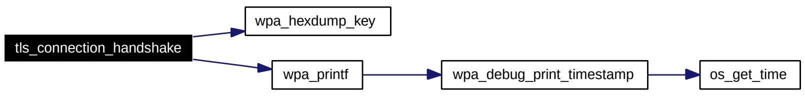
This function is used during TLS handshake. The first call is done with in_data == NULL and the library is expected to return ClientHello packet. This packet is then send to the server and a response from server is given to TLS library by calling this function again with in_data pointing to the TLS message from the server.

If the TLS handshake fails, this function may return NULL. However, if the TLS library has a TLS alert to send out, that should be returned as the output data. In this case, tls_connection_get_failed() must return failure (> 0).

tls_connection_established() should return 1 once the TLS handshake has been completed successfully.

Definition at line 930 of file tls_gnutls.c.

Here is the call graph for this function:



6.133.2.14 int tls_connection_ia_final_phase_finished (void * *tls_ctx*, struct tls_connection * *conn*)

Has final phase been completed.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

1 if valid FinalPhaseFinished has been received, 0 if not, or -1 on failure

Definition at line 1328 of file tls_gnutls.c.

6.133.2.15 int tls_connection_ia_permute_inner_secret (void * *tls_ctx*, struct tls_connection * *conn*, const u8 * *key*, size_t *key_len*)

Permute TLS/IA inner secret.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

key Session key material (session_key vectors with 2-octet length), or NULL if no session key was generating in the current phase

key_len Length of session key material

Returns:

0 on success, -1 on failure

Definition at line 1338 of file tls_gnutls.c.

6.133.2.16 int tls_connection_ia_send_phase_finished (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *final*, u8 ∗ *out_data*, size_t *out_len*)

Send a TLS/IA PhaseFinished message.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

final 1 = FinalPhaseFinished, 0 = IntermediatePhaseFinished

out_data Pointer to output buffer (encrypted TLS/IA data)

out_len Maximum out_data length

Returns:

Number of bytes written to out_data on success, -1 on failure

This function is used to send the TLS/IA end phase message, e.g., when the EAP server completes EAP-TTLSv1.

Definition at line 1280 of file tls_gnutls.c.

6.133.2.17 struct tls_connection∗ tls_connection_init (void ∗ *tls_ctx*)

Initialize a new TLS connection.

Parameters:

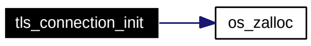
tls_ctx TLS context data from tls_init()

Returns:

Connection context data, conn for other function calls

Definition at line 325 of file tls_gnutls.c.

Here is the call graph for this function:



6.133.2.18 int tls_connection_prf (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const char ∗ *label*, int *server_random_first*, u8 ∗ *out*, size_t *out_len*)

Use TLS-PRF to derive keying material.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

label Label (e.g., description of the key) for PRF

server_random_first seed is 0 = client_random|server_random, 1 = server_random|client_random

out Buffer for output data from TLS-PRF

out_len Length of the output buffer

Returns:

0 on success, -1 on failure

This function is optional to implement if tls_connection_get_keys() provides access to master secret and server/client random values. If these values are not exported from the TLS library, tls_connection_prf() is required so that further keying material can be derived from the master secret. If not implemented, the function will still need to be defined, but it can just return -1. Example implementation of this function is in tls_prf() function when it is called with seed set to client_random|server_random (or server_-random|client_random).

Definition at line 827 of file tls_gnutls.c.

6.133.2.19 int tls_connection_resumed (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Was session resumption used.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

1 if current session used session resumption, 0 if not

Definition at line 1149 of file tls_gnutls.c.

6.133.2.20 u8∗ tls_connection_server_handshake (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, size_t ∗ *out_len*)

Process TLS handshake (server side).

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Input data from TLS peer

in_len Input data length

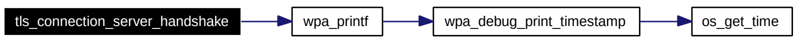
out_len Length of the output buffer.

Returns:
pointer to output data, NULL on failure

Caller is responsible for freeing returned output data.

Definition at line 1028 of file tls_gnutls.c.

Here is the call graph for this function:



6.133.2.21 int tls_connection_set_cipher_list (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, u8 ∗ *ciphers*)

Configure acceptable cipher suites.

Parameters:

tls_ctx TLS context data from tls_init()

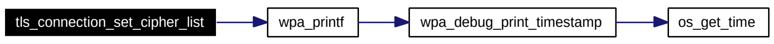
conn Connection context data from tls_connection_init()

ciphers Zero (TLS_CIPHER_NONE) terminated list of allowed ciphers (TLS_CIPHER_∗).

Returns:
0 on success, -1 on failure

Definition at line 1165 of file tls_gnutls.c.

Here is the call graph for this function:



6.133.2.22 int tls_connection_set_ia (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *tls_ia*)

Set TLS/IA parameters.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

tls_ia 1 = enable TLS/IA

Returns:
0 on success, -1 on failure

This function is used to configure TLS/IA in server mode where tls_connection_set_params() is not used.

Definition at line 1243 of file tls_gnutls.c.

6.133.2.23 int tls_connection_set_master_key (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *key*, size_t *key_len*)

Configure master secret for TLS connection.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

key TLS pre-master-secret

key_len length of key in bytes

Returns:

0 on success, -1 on failure

Definition at line 1157 of file tls_gnutls.c.

6.133.2.24 int tls_connection_set_params (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const struct tls_connection_params ∗ *params*)

Set TLS connection parameters.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

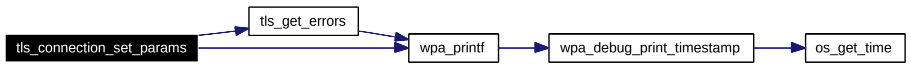
params Connection parameters

Returns:

0 on success, -1 on failure, TLS_SET_PARAMS_ENGINE_PRV_INIT_FAILED (-2) on possible PIN error causing PKCS#11 engine failure, or TLS_SET_PARAMS_ENGINE_PRV_VERIFY_FAILED (-3) on failure to verify the PKCS#11 engine private key.

Definition at line 548 of file tls_gnutls.c.

Here is the call graph for this function:



6.133.2.25 int tls_connection_set_verify (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *verify_peer*)

Set certificate verification options.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

verify_peer 1 = verify peer certificate

Returns:
0 on success, -1 on failure

Definition at line 775 of file tls_gnutls.c.

6.133.2.26 int tls_connection_shutdown (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Shutdown TLS connection.

Parameters:
tls_ctx TLS context data from tls_init()
conn Connection context data from tls_connection_init()

Returns:
0 on success, -1 on failure

Shutdown current TLS connection without releasing all resources. New connection can be started by using the same conn without having to call tls_connection_init() or setting certificates etc. again. The new connection should try to use session resumption.

Definition at line 394 of file tls_gnutls.c.

6.133.2.27 void tls_deinit (void ∗ *tls_ctx*)

Deinitialize TLS library.

Parameters:
tls_ctx TLS context data from tls_init()

Called once during program shutdown and once for each RSN pre-authentication session. If global library deinitialization is needed (i.e., one that is shared between both authentication types), the TLS library wrapper should maintain a reference counter and do global deinitialization only when moving from 1 to 0 references.

Definition at line 215 of file tls_gnutls.c.

6.133.2.28 int tls_get_cipher (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, char ∗ *buf*, size_t *buflen*)

Get current cipher name.

Parameters:
tls_ctx TLS context data from tls_init()
conn Connection context data from tls_connection_init()
buf Buffer for the cipher name
buflen buf size

Returns:
0 on success, -1 on failure

Get the name of the currently used cipher.

Definition at line 1173 of file tls_gnutls.c.

6.133.2.29 int tls_get_errors (void ∗ *tls_ctx*)

Process pending errors.

Parameters:

tls_ctx TLS context data from tls_init()

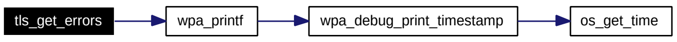
Returns:

Number of found error, 0 if no errors detected.

Process all pending TLS errors.

Definition at line 231 of file tls_gnutls.c.

Here is the call graph for this function:



6.133.2.30 int tls_global_set_params (void ∗ *tls_ctx*, const struct tls_connection_params ∗ *params*)

Set TLS parameters for all TLS connection.

Parameters:

tls_ctx TLS context data from tls_init()

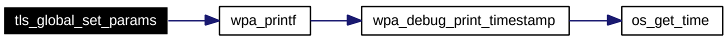
params Global TLS parameters

Returns:

0 on success, -1 on failure, TLS_SET_PARAMS_ENGINE_PRV_INIT_FAILED (-2) on possible PIN error causing PKCS#11 engine failure, or TLS_SET_PARAMS_ENGINE_PRV_VERIFY_FAILED (-3) on failure to verify the PKCS#11 engine private key.

Definition at line 674 of file tls_gnutls.c.

Here is the call graph for this function:



6.133.2.31 int tls_global_set_verify (void ∗ *tls_ctx*, int *check_crl*)

Set global certificate verification options.

Parameters:

tls_ctx TLS context data from tls_init()

check_crl 0 = do not verify CRLs, 1 = verify CRL for the user certificate, 2 = verify CRL for all certificates

Returns:

0 on success, -1 on failure

Definition at line 768 of file tls_gnutls.c.

6.133.2.32 void∗ tls_init (const struct tls_config ∗ *conf*)

Initialize TLS library.

Parameters:

conf Configuration data for TLS library

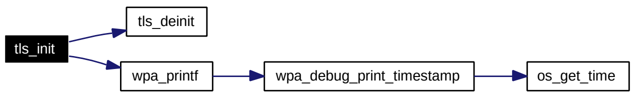
Returns:

Context data to be used as tls_ctx in calls to other functions, or NULL on failure.

Called once during program startup and once for each RSN pre-authentication session. In other words, there can be two concurrent TLS contexts. If global library initialization is needed (i.e., one that is shared between both authentication types), the TLS library wrapper should maintain a reference counter and do global initialization only when moving from 0 to 1 reference.

Definition at line 163 of file tls_gnutls.c.

Here is the call graph for this function:

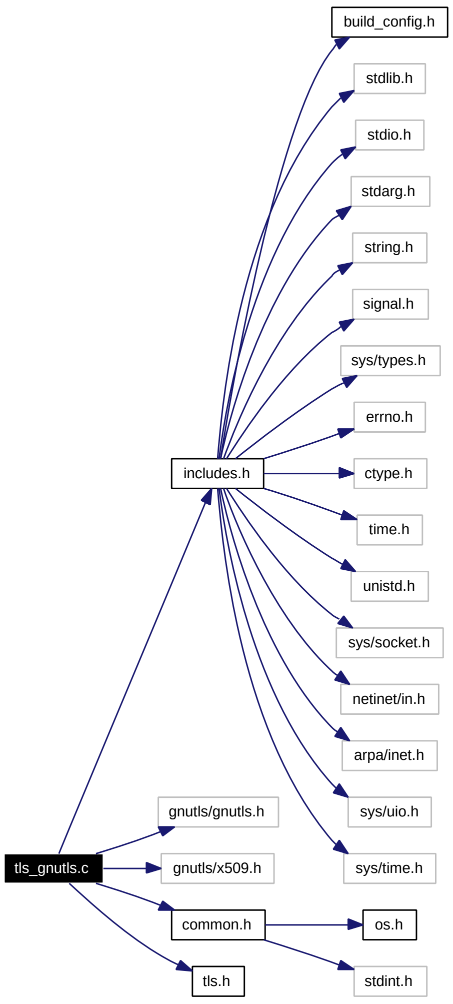


6.134 tls_gnutls.c File Reference

WPA Supplicant / SSL/TLS interface functions for openssl.

```
#include "includes.h"
#include <gnutls/gnutls.h>
#include <gnutls/x509.h>
#include "common.h"
#include "tls.h"
```

Include dependency graph for tls_gnutls.c:



Defines

- #define **TLS_RANDOM_SIZE** 32
- #define **TLS_MASTER_SIZE** 48
- #define **GNUTLS_INTERNAL_STRUCTURE_HACK**

Typedefs

- typedef u8 **uint8**
- typedef unsigned char **opaque**

Functions

- void ∗ tls_init (const struct tls_config ∗conf)

  Initialize TLS library.

- void tls_deinit (void ∗ssl_ctx)

  Deinitialize TLS library.

- int tls_get_errors (void ∗ssl_ctx)

  Process pending errors.

- tls_connection ∗ tls_connection_init (void ∗ssl_ctx)

  Initialize a new TLS connection.

- void tls_connection_deinit (void ∗ssl_ctx, struct tls_connection ∗conn)

  Free TLS connection data.

- int tls_connection_established (void ∗ssl_ctx, struct tls_connection ∗conn)

  Has the TLS connection been completed?

- int tls_connection_shutdown (void ∗ssl_ctx, struct tls_connection ∗conn)

  Shutdown TLS connection.

- int tls_connection_set_params (void ∗tls_ctx, struct tls_connection ∗conn, const struct tls_-connection_params ∗params)

  Set TLS connection parameters.

- int tls_global_set_params (void ∗tls_ctx, const struct tls_connection_params ∗params)

  Set TLS parameters for all TLS connection.

- int tls_global_set_verify (void ∗ssl_ctx, int check_crl)

  Set global certificate verification options.

- int tls_connection_set_verify (void ∗ssl_ctx, struct tls_connection ∗conn, int verify_peer)

  Set certificate verification options.

- int tls_connection_get_keys (void ∗ssl_ctx, struct tls_connection ∗conn, struct tls_keys ∗keys)

  Get master key and random data from TLS connection.

- int tls_connection_prf (void ∗tls_ctx, struct tls_connection ∗conn, const char ∗label, int server_-random_first, u8 ∗out, size_t out_len)

  Use TLS-PRF to derive keying material.

- u8 ∗ tls_connection_handshake (void ∗ssl_ctx, struct tls_connection ∗conn, const u8 ∗in_data, size_t in_len, size_t ∗out_len, u8 ∗∗appl_data, size_t ∗appl_data_len)

Process TLS handshake (client side).

- u8 ∗ tls_connection_server_handshake (void ∗ssl_ctx, struct tls_connection ∗conn, const u8 ∗in_-data, size_t in_len, size_t ∗out_len)

  Process TLS handshake (server side).

- int tls_connection_encrypt (void ∗ssl_ctx, struct tls_connection ∗conn, const u8 ∗in_data, size_t in_-len, u8 ∗out_data, size_t out_len)

  Encrypt data into TLS tunnel.

- int tls_connection_decrypt (void ∗ssl_ctx, struct tls_connection ∗conn, const u8 ∗in_data, size_t in_-len, u8 ∗out_data, size_t out_len)

  Decrypt data from TLS tunnel.

- int tls_connection_resumed (void ∗ssl_ctx, struct tls_connection ∗conn)

  Was session resumption used.

- int tls_connection_set_master_key (void ∗ssl_ctx, struct tls_connection ∗conn, const u8 ∗key, size_t key_len)

  Configure master secret for TLS connection.

- int tls_connection_set_cipher_list (void ∗tls_ctx, struct tls_connection ∗conn, u8 ∗ciphers)

  Configure acceptable cipher suites.

- int tls_get_cipher (void ∗ssl_ctx, struct tls_connection ∗conn, char ∗buf, size_t buflen)

  Get current cipher name.

- int tls_connection_enable_workaround (void ∗ssl_ctx, struct tls_connection ∗conn)

  Enable TLS workaround options.

- int tls_connection_client_hello_ext (void ∗ssl_ctx, struct tls_connection ∗conn, int ext_type, const u8 ∗data, size_t data_len)

  Set TLS extension for ClientHello.

- int tls_connection_get_failed (void ∗ssl_ctx, struct tls_connection ∗conn)

  Get connection failure status.

- int tls_connection_get_read_alerts (void ∗ssl_ctx, struct tls_connection ∗conn)

  Get connection read alert status.

- int tls_connection_get_write_alerts (void ∗ssl_ctx, struct tls_connection ∗conn)

  Get connection write alert status.

- int tls_connection_get_keyblock_size (void ∗tls_ctx, struct tls_connection ∗conn)

  Get TLS key_block size.

- unsigned int tls_capabilities (void ∗tls_ctx)

  Get supported TLS capabilities.

- int tls_connection_set_ia (void ∗tls_ctx, struct tls_connection ∗conn, int tls_ia)

Set TLS/IA parameters.

- int tls_connection_ia_send_phase_finished (void ∗tls_ctx, struct tls_connection ∗conn, int final, u8 ∗out_data, size_t out_len)

  Send a TLS/IA PhaseFinished message.

- int tls_connection_ia_final_phase_finished (void ∗tls_ctx, struct tls_connection ∗conn)

  Has final phase been completed.

- int tls_connection_ia_permute_inner_secret (void ∗tls_ctx, struct tls_connection ∗conn, const u8 ∗key, size_t key_len)

  Permute TLS/IA inner secret.

Variables

- int **wpa_debug_show_keys**

6.134.1 Detailed Description

WPA Supplicant / SSL/TLS interface functions for openssl.

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file tls_gnutls.c.

6.134.2 Function Documentation

6.134.2.1 unsigned int tls_capabilities (void ∗ *tls_ctx*)

Get supported TLS capabilities.

Parameters:

tls_ctx TLS context data from tls_init()

Returns:

Bit field of supported TLS capabilities (TLS_CAPABILITY_∗)

Definition at line 1231 of file tls_gnutls.c.

6.134.2.2 int tls_connection_client_hello_ext (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *ext_type*, const u8 ∗ *data*, size_t *data_len*)

Set TLS extension for ClientHello.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

ext_type Extension type

data Extension payload (NULL to remove extension)

data_len Extension payload length

Returns:

0 on success, -1 on failure

Definition at line 1190 of file tls_gnutls.c.

6.134.2.3 int tls_connection_decrypt (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, u8 ∗ *out_data*, size_t *out_len*)

Decrypt data from TLS tunnel.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Pointer to input buffer (encrypted TLS data)

in_len Input buffer length

out_data Pointer to output buffer (decrypted data from TLS tunnel)

out_len Maximum out_data length

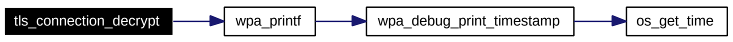
Returns:

Number of bytes written to out_data, -1 on failure

This function is used after TLS handshake has been completed successfully to receive data from the encrypted tunnel.

Definition at line 1067 of file tls_gnutls.c.

Here is the call graph for this function:



6.134.2.4 void tls_connection_deinit (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Free TLS connection data.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Release all resources allocated for TLS connection.

Definition at line 361 of file tls_gnutls.c.

6.134.2.5 int tls_connection_enable_workaround (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Enable TLS workaround options.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

0 on success, -1 on failure

This function is used to enable connection-specific workaround options for buffer SSL/TLS implementations.

Definition at line 1182 of file tls_gnutls.c.

6.134.2.6 int tls_connection_encrypt (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, u8 ∗ *out_data*, size_t *out_len*)

Encrypt data into TLS tunnel.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Pointer to plaintext data to be encrypted

in_len Input buffer length

out_data Pointer to output buffer (encrypted TLS data)

out_len Maximum out_data length

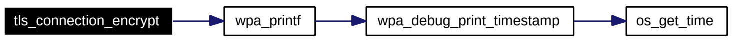
Returns:

Number of bytes written to out_data, -1 on failure

This function is used after TLS handshake has been completed successfully to send data in the encrypted tunnel.

Definition at line 1038 of file tls_gnutls.c.

Here is the call graph for this function:



6.134.2.7 int tls_connection_established (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Has the TLS connection been completed?

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

1 if TLS connection has been completed, 0 if not.

Definition at line 388 of file tls_gnutls.c.

6.134.2.8 int tls_connection_get_failed (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get connection failure status.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns >0 if connection has failed, 0 if not.

Definition at line 1199 of file tls_gnutls.c.

6.134.2.9 int tls_connection_get_keyblock_size (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get TLS key_block size.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

Size of the key_block for the negotiated cipher suite or -1 on failure

Definition at line 1223 of file tls_gnutls.c.

6.134.2.10 int tls_connection_get_keys (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, struct tls_keys ∗ *keys*)

Get master key and random data from TLS connection.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

keys Structure of key/random data (filled on success)

Returns:

0 on success, -1 on failure

Definition at line 790 of file tls_gnutls.c.

6.134.2.11 int tls_connection_get_read_alerts (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get connection read alert status.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

Number of times a fatal read (remote end reported error) has happened during this connection.

Definition at line 1207 of file tls_gnutls.c.

6.134.2.12 int tls_connection_get_write_alerts (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get connection write alert status.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

Number of times a fatal write (locally detected error) has happened during this connection.

Definition at line 1215 of file tls_gnutls.c.

6.134.2.13 u8∗ tls_connection_handshake (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, size_t ∗ *out_len*, u8 ∗∗ *appl_data*, size_t ∗ *appl_data_len*)

Process TLS handshake (client side).

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Input data from TLS peer

in_len Input data length

out_len Length of the output buffer.

appl_data Pointer to application data pointer, or NULL if dropped

appl_data_len Pointer to variable that is set to appl_data length

Returns:

Pointer to output data, NULL on failure

Caller is responsible for freeing returned output data. If the final handshake message includes application data, this is decrypted and appl_data (if not NULL) is set to point this data. Caller is responsible for freeing appl_data.

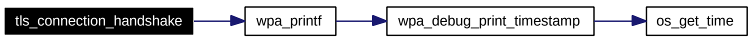
This function is used during TLS handshake. The first call is done with in_data == NULL and the library is expected to return ClientHello packet. This packet is then send to the server and a response from server is given to TLS library by calling this function again with in_data pointing to the TLS message from the server.

If the TLS handshake fails, this function may return NULL. However, if the TLS library has a TLS alert to send out, that should be returned as the output data. In this case, tls_connection_get_failed() must return failure (> 0).

tls_connection_established() should return 1 once the TLS handshake has been completed successfully.

Definition at line 930 of file tls_gnutls.c.

Here is the call graph for this function:



6.134.2.14 int tls_connection_ia_final_phase_finished (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Has final phase been completed.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

1 if valid FinalPhaseFinished has been received, 0 if not, or -1 on failure

Definition at line 1328 of file tls_gnutls.c.

6.134.2.15 int tls_connection_ia_permute_inner_secret (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *key*, size_t *key_len*)

Permute TLS/IA inner secret.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

key Session key material (session_key vectors with 2-octet length), or NULL if no session key was generating in the current phase

key_len Length of session key material

Returns:

0 on success, -1 on failure

Definition at line 1338 of file tls_gnutls.c.

6.134.2.16 int tls_connection_ia_send_phase_finished (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *final*, u8 ∗ *out_data*, size_t *out_len*)

Send a TLS/IA PhaseFinished message.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

final 1 = FinalPhaseFinished, 0 = IntermediatePhaseFinished

out_data Pointer to output buffer (encrypted TLS/IA data)

out_len Maximum out_data length

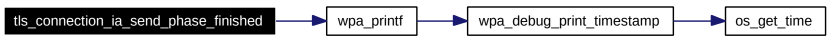
Returns:

Number of bytes written to out_data on success, -1 on failure

This function is used to send the TLS/IA end phase message, e.g., when the EAP server completes EAP-TTLSv1.

Definition at line 1280 of file tls_gnutls.c.

Here is the call graph for this function:


tls_connection_ia_send_phase_finished
wpa_printf
wpa_debug_print_timestamp
os_get_time

6.134.2.17 struct tls_connection∗ tls_connection_init (void ∗ *tls_ctx*)

Initialize a new TLS connection.

Parameters:

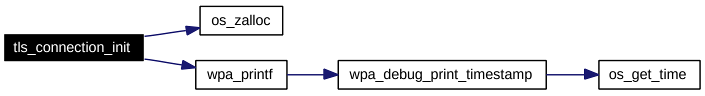
tls_ctx TLS context data from tls_init()

Returns:

Connection context data, conn for other function calls

Definition at line 325 of file tls_gnutls.c.

Here is the call graph for this function:



6.134.2.18 int tls_connection_prf (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const char ∗ *label*, int *server_random_first*, u8 ∗ *out*, size_t *out_len*)

Use TLS-PRF to derive keying material.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

label Label (e.g., description of the key) for PRF

server_random_first seed is 0 = client_random|server_random, 1 = server_random|client_random

out Buffer for output data from TLS-PRF

out_len Length of the output buffer

Returns:

0 on success, -1 on failure

This function is optional to implement if tls_connection_get_keys() provides access to master secret and server/client random values. If these values are not exported from the TLS library, tls_connection_prf() is required so that further keying material can be derived from the master secret. If not implemented, the function will still need to be defined, but it can just return -1. Example implementation of this function is in tls_prf() function when it is called with seed set to client_random|server_random (or server_-random|client_random).

Definition at line 827 of file tls_gnutls.c.

6.134.2.19 int tls_connection_resumed (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Was session resumption used.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

1 if current session used session resumption, 0 if not

Definition at line 1149 of file tls_gnutls.c.

6.134.2.20 u8∗ tls_connection_server_handshake (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, size_t ∗ *out_len*)

Process TLS handshake (server side).

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Input data from TLS peer

in_len Input data length

out_len Length of the output buffer.

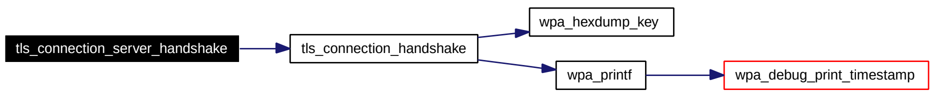
Returns:

pointer to output data, NULL on failure

Caller is responsible for freeing returned output data.

Definition at line 1028 of file tls_gnutls.c.

Here is the call graph for this function:



6.134.2.21 int tls_connection_set_cipher_list (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, u8 ∗ *ciphers*)

Configure acceptable cipher suites.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

ciphers Zero (TLS_CIPHER_NONE) terminated list of allowed ciphers (TLS_CIPHER_∗).

Returns:

0 on success, -1 on failure

Definition at line 1165 of file tls_gnutls.c.

6.134.2.22 int tls_connection_set_ia (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *tls_ia*)

Set TLS/IA parameters.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

tls_ia 1 = enable TLS/IA

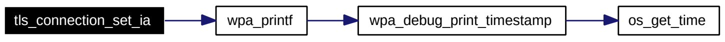
Returns:

0 on success, -1 on failure

This function is used to configure TLS/IA in server mode where tls_connection_set_params() is not used.

Definition at line 1243 of file tls_gnutls.c.

Here is the call graph for this function:



6.134.2.23 int tls_connection_set_master_key (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *key*, size_t *key_len*)

Configure master secret for TLS connection.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

key TLS pre-master-secret

key_len length of key in bytes

Returns:

0 on success, -1 on failure

Definition at line 1157 of file tls_gnutls.c.

6.134.2.24 int tls_connection_set_params (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const struct tls_connection_params ∗ *params*)

Set TLS connection parameters.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

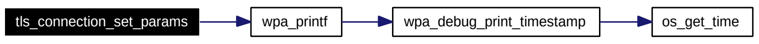
params Connection parameters

Returns:

0 on success, -1 on failure, TLS_SET_PARAMS_ENGINE_PRV_INIT_FAILED (-2) on possible PIN error causing PKCS#11 engine failure, or TLS_SET_PARAMS_ENGINE_PRV_VERIFY_FAILED (-3) on failure to verify the PKCS#11 engine private key.

Definition at line 548 of file tls_gnutls.c.

Here is the call graph for this function:



6.134.2.25 int tls_connection_set_verify (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *verify_peer*)

Set certificate verification options.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

verify_peer 1 = verify peer certificate

Returns:

0 on success, -1 on failure

Definition at line 775 of file tls_gnutls.c.

6.134.2.26 int tls_connection_shutdown (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Shutdown TLS connection.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

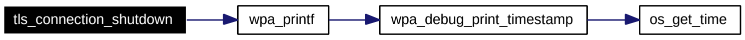
Returns:

0 on success, -1 on failure

Shutdown current TLS connection without releasing all resources. New connection can be started by using the same conn without having to call tls_connection_init() or setting certificates etc. again. The new connection should try to use session resumption.

Definition at line 394 of file tls_gnutls.c.

Here is the call graph for this function:



6.134.2.27 void tls_deinit (void ∗ *tls_ctx*)

Deinitialize TLS library.

Parameters:

tls_ctx TLS context data from tls_init()

Called once during program shutdown and once for each RSN pre-authentication session. If global library deinitialization is needed (i.e., one that is shared between both authentication types), the TLS library wrapper should maintain a reference counter and do global deinitialization only when moving from 1 to 0 references.

Definition at line 215 of file tls_gnutls.c.

6.134.2.28 int tls_get_cipher (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, char ∗ *buf*, size_t *buflen*)

Get current cipher name.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

buf Buffer for the cipher name

buflen buf size

Returns:

0 on success, -1 on failure

Get the name of the currently used cipher.

Definition at line 1173 of file tls_gnutls.c.

6.134.2.29 int tls_get_errors (void ∗ *tls_ctx*)

Process pending errors.

Parameters:

tls_ctx TLS context data from tls_init()

Returns:

Number of found error, 0 if no errors detected.

Process all pending TLS errors.

Definition at line 231 of file tls_gnutls.c.

6.134.2.30 int tls_global_set_params (void ∗ *tls_ctx*, const struct tls_connection_params ∗ *params*)

Set TLS parameters for all TLS connection.

Parameters:

tls_ctx TLS context data from tls_init()

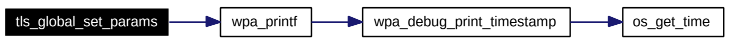
params Global TLS parameters

Returns:

0 on success, -1 on failure, TLS_SET_PARAMS_ENGINE_PRV_INIT_FAILED (-2) on possible PIN error causing PKCS#11 engine failure, or TLS_SET_PARAMS_ENGINE_PRV_VERIFY_FAILED (-3) on failure to verify the PKCS#11 engine private key.

Definition at line 674 of file tls_gnutls.c.

Here is the call graph for this function:



6.134.2.31 int tls_global_set_verify (void ∗ *tls_ctx*, int *check_crl*)

Set global certificate verification options.

Parameters:

tls_ctx TLS context data from tls_init()

check_crl 0 = do not verify CRLs, 1 = verify CRL for the user certificate, 2 = verify CRL for all certificates

Returns:

0 on success, -1 on failure

Definition at line 768 of file tls_gnutls.c.

6.134.2.32 void∗ tls_init (const struct tls_config ∗ *conf*)

Initialize TLS library.

Parameters:

conf Configuration data for TLS library

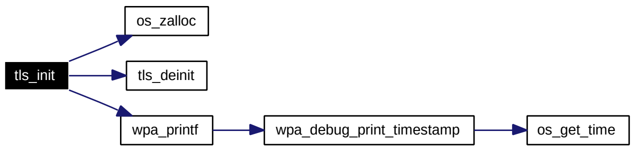
Returns:

Context data to be used as tls_ctx in calls to other functions, or NULL on failure.

Called once during program startup and once for each RSN pre-authentication session. In other words, there can be two concurrent TLS contexts. If global library initialization is needed (i.e., one that is shared between both authentication types), the TLS library wrapper should maintain a reference counter and do global initialization only when moving from 0 to 1 reference.

Definition at line 163 of file tls_gnutls.c.

Here is the call graph for this function:

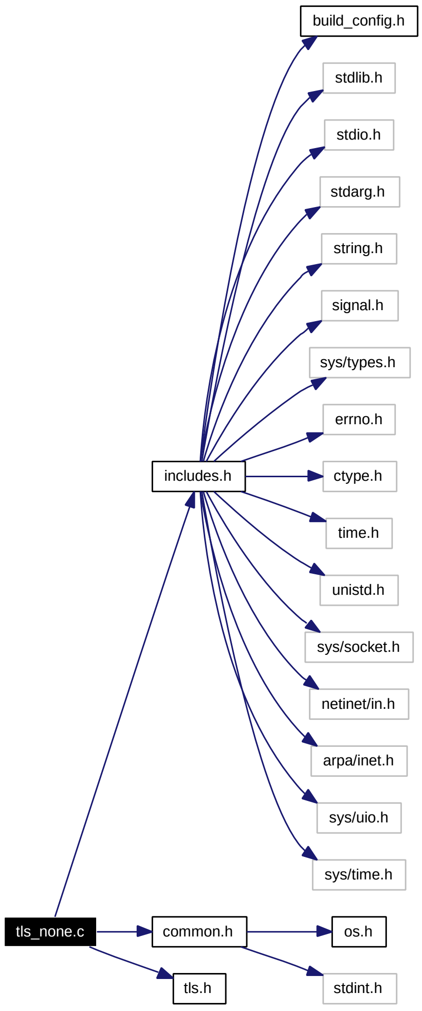


6.135 tls_none.c File Reference

WPA Supplicant / SSL/TLS interface functions for no TLS case.

```
#include "includes.h"
#include "common.h"
#include "tls.h"
```

Include dependency graph for tls_none.c:



Functions

- void ∗ tls_init (const struct tls_config ∗conf)

  Initialize TLS library.

- void tls_deinit (void ∗ssl_ctx)

  Deinitialize TLS library.

6.135.1 Detailed Description

WPA Supplicant / SSL/TLS interface functions for no TLS case.

Copyright

Copyright (c) 2004, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file tls_none.c.

6.135.2 Function Documentation

6.135.2.1 void tls_deinit (void ∗ *tls_ctx*)

Deinitialize TLS library.

Parameters:

tls_ctx TLS context data from tls_init()

Called once during program shutdown and once for each RSN pre-authentication session. If global library deinitialization is needed (i.e., one that is shared between both authentication types), the TLS library wrapper should maintain a reference counter and do global deinitialization only when moving from 1 to 0 references.

Definition at line 26 of file tls_none.c.

6.135.2.2 void∗ tls_init (const struct tls_config ∗ *conf*)

Initialize TLS library.

Parameters:

conf Configuration data for TLS library

Returns:

Context data to be used as tls_ctx in calls to other functions, or NULL on failure.

Called once during program startup and once for each RSN pre-authentication session. In other words, there can be two concurrent TLS contexts. If global library initialization is needed (i.e., one that is shared between both authentication types), the TLS library wrapper should maintain a reference counter and do global initialization only when moving from 0 to 1 reference.

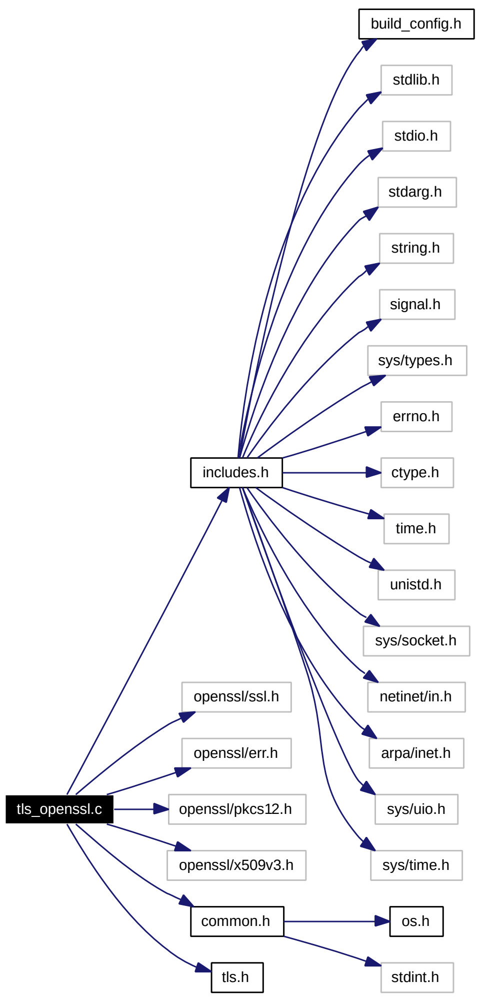
Definition at line 21 of file tls_none.c.

6.136 tls_openssl.c File Reference

WPA Supplicant / SSL/TLS interface functions for openssl.

```
#include "includes.h"
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <openssl/pkcs12.h>
#include <openssl/x509v3.h>
#include "common.h"
#include "tls.h"
```

Include dependency graph for tls_openssl.c:



Data Structures

- struct **tls_connection**

Defines

- #define **OPENSSL_d2i_TYPE** unsigned char ∗∗

Functions

- void ∗ tls_init (const struct tls_config ∗conf)

  Initialize TLS library.

- void tls_deinit (void ∗ssl_ctx)

  Deinitialize TLS library.

- int tls_get_errors (void ∗ssl_ctx)

  Process pending errors.

- tls_connection ∗ tls_connection_init (void ∗ssl_ctx)

  Initialize a new TLS connection.

- void tls_connection_deinit (void ∗ssl_ctx, struct tls_connection ∗conn)

  Free TLS connection data.

- int tls_connection_established (void ∗ssl_ctx, struct tls_connection ∗conn)

  Has the TLS connection been completed?

- int tls_connection_shutdown (void ∗ssl_ctx, struct tls_connection ∗conn)

  Shutdown TLS connection.

- int tls_global_set_verify (void ∗ssl_ctx, int check_crl)

  Set global certificate verification options.

- int tls_connection_set_verify (void ∗ssl_ctx, struct tls_connection ∗conn, int verify_peer)

  Set certificate verification options.

- int tls_connection_get_keys (void ∗ssl_ctx, struct tls_connection ∗conn, struct tls_keys ∗keys)

  Get master key and random data from TLS connection.

- int tls_connection_prf (void ∗tls_ctx, struct tls_connection ∗conn, const char ∗label, int server_-random_first, u8 ∗out, size_t out_len)

  Use TLS-PRF to derive keying material.

- u8 ∗ tls_connection_handshake (void ∗ssl_ctx, struct tls_connection ∗conn, const u8 ∗in_data, size_t in_len, size_t ∗out_len, u8 ∗∗appl_data, size_t ∗appl_data_len)

  Process TLS handshake (client side).

- u8 ∗ tls_connection_server_handshake (void ∗ssl_ctx, struct tls_connection ∗conn, const u8 ∗in_-data, size_t in_len, size_t ∗out_len)

  Process TLS handshake (server side).

- int tls_connection_encrypt (void ∗ssl_ctx, struct tls_connection ∗conn, const u8 ∗in_data, size_t in_-len, u8 ∗out_data, size_t out_len)

Encrypt data into TLS tunnel.

- int tls_connection_decrypt (void ∗ssl_ctx, struct tls_connection ∗conn, const u8 ∗in_data, size_t in_-len, u8 ∗out_data, size_t out_len)

  Decrypt data from TLS tunnel.

- int tls_connection_resumed (void ∗ssl_ctx, struct tls_connection ∗conn)

  Was session resumption used.

- int tls_connection_set_cipher_list (void ∗tls_ctx, struct tls_connection ∗conn, u8 ∗ciphers)

  Configure acceptable cipher suites.

- int tls_get_cipher (void ∗ssl_ctx, struct tls_connection ∗conn, char ∗buf, size_t buflen)

  Get current cipher name.

- int tls_connection_enable_workaround (void ∗ssl_ctx, struct tls_connection ∗conn)

  Enable TLS workaround options.

- int tls_connection_get_failed (void ∗ssl_ctx, struct tls_connection ∗conn)

  Get connection failure status.

- int tls_connection_get_read_alerts (void ∗ssl_ctx, struct tls_connection ∗conn)

  Get connection read alert status.

- int tls_connection_get_write_alerts (void ∗ssl_ctx, struct tls_connection ∗conn)

  Get connection write alert status.

- int tls_connection_set_params (void ∗tls_ctx, struct tls_connection ∗conn, const struct tls_-connection_params ∗params)

  Set TLS connection parameters.

- int tls_global_set_params (void ∗tls_ctx, const struct tls_connection_params ∗params)

  Set TLS parameters for all TLS connection.

- int tls_connection_get_keyblock_size (void ∗tls_ctx, struct tls_connection ∗conn)

  Get TLS key_block size.

- unsigned int tls_capabilities (void ∗tls_ctx)

  Get supported TLS capabilities.

- int tls_connection_set_ia (void ∗tls_ctx, struct tls_connection ∗conn, int tls_ia)

  Set TLS/IA parameters.

- int tls_connection_ia_send_phase_finished (void ∗tls_ctx, struct tls_connection ∗conn, int final, u8 ∗out_data, size_t out_len)

  Send a TLS/IA PhaseFinished message.

- int tls_connection_ia_final_phase_finished (void ∗tls_ctx, struct tls_connection ∗conn)

  Has final phase been completed.

- int tls_connection_ia_permute_inner_secret (void ∗tls_ctx, struct tls_connection ∗conn, const u8 ∗key, size_t key_len)

    Permute TLS/IA inner secret.

6.136.1 Detailed Description

WPA Supplicant / SSL/TLS interface functions for openssl.

Copyright

Copyright (c) 2004-2006, Jouni Malinen <jkmaline@cc.hut.fi>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file tls_openssl.c.

6.136.2 Function Documentation

6.136.2.1 unsigned int tls_capabilities (void ∗ *tls_ctx*)

Get supported TLS capabilities.

Parameters:

tls_ctx TLS context data from tls_init()

Returns:

Bit field of supported TLS capabilities (TLS_CAPABILITY_∗)

Definition at line 2278 of file tls_openssl.c.

6.136.2.2 int tls_connection_decrypt (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, u8 ∗ *out_data*, size_t *out_len*)

Decrypt data from TLS tunnel.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Pointer to input buffer (encrypted TLS data)

in_len Input buffer length

out_data Pointer to output buffer (decrypted data from TLS tunnel)

out_len Maximum out_data length

Returns:

Number of bytes written to out_data, -1 on failure

This function is used after TLS handshake has been completed successfully to receive data from the encrypted tunnel.

Definition at line 1961 of file tls_openssl.c.

6.136.2.3 void tls_connection_deinit (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Free TLS connection data.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Release all resources allocated for TLS connection.

Definition at line 920 of file tls_openssl.c.

6.136.2.4 int tls_connection_enable_workaround (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Enable TLS workaround options.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

0 on success, -1 on failure

This function is used to enable connection-specific workaround options for buffer SSL/TLS implementations.

Definition at line 2121 of file tls_openssl.c.

6.136.2.5 int tls_connection_encrypt (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, u8 ∗ *out_data*, size_t *out_len*)

Encrypt data into TLS tunnel.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Pointer to plaintext data to be encrypted

in_len Input buffer length

out_data Pointer to output buffer (encrypted TLS data)

out_len Maximum out_data length

Returns:

Number of bytes written to out_data, -1 on failure

This function is used after TLS handshake has been completed successfully to send data in the encrypted tunnel.

Definition at line 1927 of file tls_openssl.c.

6.136.2.6 int tls_connection_established (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Has the TLS connection been completed?

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

1 if TLS connection has been completed, 0 if not.

Definition at line 933 of file tls_openssl.c.

6.136.2.7 int tls_connection_get_failed (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get connection failure status.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns >0 if connection has failed, 0 if not.

Definition at line 2150 of file tls_openssl.c.

6.136.2.8 int tls_connection_get_keyblock_size (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get TLS key_block size.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

Size of the key_block for the negotiated cipher suite or -1 on failure

Definition at line 2257 of file tls_openssl.c.

6.136.2.9 int tls_connection_get_keys (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, struct tls_keys ∗ *keys*)

Get master key and random data from TLS connection.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

keys Structure of key/random data (filled on success)

Returns:

0 on success, -1 on failure

Definition at line 1757 of file tls_openssl.c.

6.136.2.10 int tls_connection_get_read_alerts (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get connection read alert status.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

Number of times a fatal read (remote end reported error) has happened during this connection.

Definition at line 2158 of file tls_openssl.c.

6.136.2.11 int tls_connection_get_write_alerts (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Get connection write alert status.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

Number of times a fatal write (locally detected error) has happened during this connection.

Definition at line 2166 of file tls_openssl.c.

6.136.2.12 u8∗ tls_connection_handshake (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, size_t ∗ *out_len*, u8 ∗∗ *appl_data*, size_t ∗ *appl_data_len*)

Process TLS handshake (client side).

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Input data from TLS peer

in_len Input data length

out_len Length of the output buffer.

appl_data Pointer to application data pointer, or NULL if dropped

appl_data_len Pointer to variable that is set to appl_data length

Returns:

Pointer to output data, NULL on failure

Caller is responsible for freeing returned output data. If the final handshake message includes application data, this is decrypted and appl_data (if not NULL) is set to point this data. Caller is responsible for freeing appl_data.

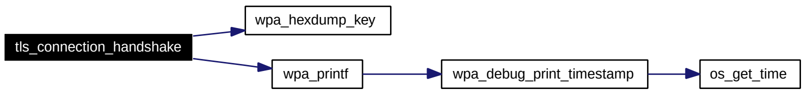
This function is used during TLS handshake. The first call is done with in_data == NULL and the library is expected to return ClientHello packet. This packet is then send to the server and a response from server is given to TLS library by calling this function again with in_data pointing to the TLS message from the server.

If the TLS handshake fails, this function may return NULL. However, if the TLS library has a TLS alert to send out, that should be returned as the output data. In this case, tls_connection_get_failed() must return failure (> 0).

tls_connection_established() should return 1 once the TLS handshake has been completed successfully.

Definition at line 1788 of file tls_openssl.c.

Here is the call graph for this function:



6.136.2.13 int tls_connection_ia_final_phase_finished (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Has final phase been completed.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

1 if valid FinalPhaseFinished has been received, 0 if not, or -1 on failure

Definition at line 2300 of file tls_openssl.c.

6.136.2.14 int tls_connection_ia_permute_inner_secret (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *key*, size_t *key_len*)

Permute TLS/IA inner secret.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

key Session key material (session_key vectors with 2-octet length), or NULL if no session key was generating in the current phase

key_len Length of session key material

Returns:

0 on success, -1 on failure

Definition at line 2307 of file tls_openssl.c.

6.136.2.15 int tls_connection_ia_send_phase_finished (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *final*, u8 ∗ *out_data*, size_t *out_len*)

Send a TLS/IA PhaseFinished message.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

final 1 = FinalPhaseFinished, 0 = IntermediatePhaseFinished

out_data Pointer to output buffer (encrypted TLS/IA data)

out_len Maximum out_data length

Returns:

Number of bytes written to out_data on success, -1 on failure

This function is used to send the TLS/IA end phase message, e.g., when the EAP server completes EAP-TTLSv1.

Definition at line 2291 of file tls_openssl.c.

6.136.2.16 struct tls_connection∗ tls_connection_init (void ∗ *tls_ctx*)

Initialize a new TLS connection.

Parameters:

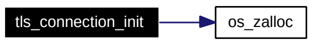
tls_ctx TLS context data from tls_init()

Returns:

Connection context data, conn for other function calls

Definition at line 874 of file tls_openssl.c.

Here is the call graph for this function:



6.136.2.17 int tls_connection_prf (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const char ∗ *label*, int *server_random_first*, u8 ∗ *out*, size_t *out_len*)

Use TLS-PRF to derive keying material.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

label Label (e.g., description of the key) for PRF

server_random_first seed is 0 = client_random|server_random, 1 = server_random|client_random

out Buffer for output data from TLS-PRF *out_len* Length of the output buffer

Returns:

0 on success, -1 on failure

This function is optional to implement if tls_connection_get_keys() provides access to master secret and server/client random values. If these values are not exported from the TLS library, tls_connection_prf() is required so that further keying material can be derived from the master secret. If not implemented, the function will still need to be defined, but it can just return -1. Example implementation of this function is in tls_prf() function when it is called with seed set to client_random|server_random (or server_-random|client_random).

Definition at line 1780 of file tls_openssl.c.

6.136.2.18 int tls_connection_resumed (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Was session resumption used.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

1 if current session used session resumption, 0 if not

Definition at line 1991 of file tls_openssl.c.

6.136.2.19 u8∗ tls_connection_server_handshake (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const u8 ∗ *in_data*, size_t *in_len*, size_t ∗ *out_len*)

Process TLS handshake (server side).

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

in_data Input data from TLS peer

in_len Input data length

out_len Length of the output buffer.

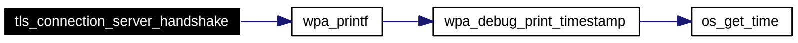
Returns:

pointer to output data, NULL on failure

Caller is responsible for freeing returned output data.

Definition at line 1876 of file tls_openssl.c.

Here is the call graph for this function:



6.136.2.20 int tls_connection_set_cipher_list (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, u8 ∗ *ciphers*)

Configure acceptable cipher suites.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

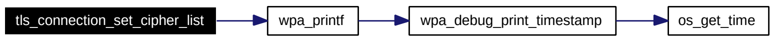
ciphers Zero (TLS_CIPHER_NONE) terminated list of allowed ciphers (TLS_CIPHER_∗).

Returns:

0 on success, -1 on failure

Definition at line 2048 of file tls_openssl.c.

Here is the call graph for this function:



6.136.2.21 int tls_connection_set_ia (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *tls_ia*)

Set TLS/IA parameters.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

tls_ia 1 = enable TLS/IA

Returns:

0 on success, -1 on failure

This function is used to configure TLS/IA in server mode where tls_connection_set_params() is not used.

Definition at line 2284 of file tls_openssl.c.

6.136.2.22 int tls_connection_set_params (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, const struct tls_connection_params ∗ *params*)

Set TLS connection parameters.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

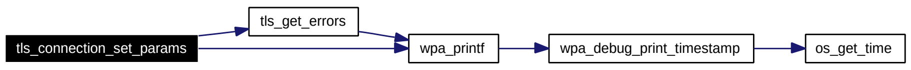
params Connection parameters

Returns:

0 on success, -1 on failure, TLS_SET_PARAMS_ENGINE_PRV_INIT_FAILED (-2) on possible PIN error causing PKCS#11 engine failure, or TLS_SET_PARAMS_ENGINE_PRV_VERIFY_FAILED (-3) on failure to verify the PKCS#11 engine private key.

Definition at line 2174 of file tls_openssl.c.

Here is the call graph for this function:



6.136.2.23 int tls_connection_set_verify (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, int *verify_peer*)

Set certificate verification options.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

verify_peer 1 = verify peer certificate

Returns:

0 on success, -1 on failure

Definition at line 1238 of file tls_openssl.c.

6.136.2.24 int tls_connection_shutdown (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*)

Shutdown TLS connection.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

Returns:

0 on success, -1 on failure

Shutdown current TLS connection without releasing all resources. New connection can be started by using the same conn without having to call tls_connection_init() or setting certificates etc. again. The new connection should try to use session resumption.

Definition at line 939 of file tls_openssl.c.

6.136.2.25 void tls_deinit (void ∗ *tls_ctx*)

Deinitialize TLS library.

Parameters:

tls_ctx TLS context data from tls_init()

Called once during program shutdown and once for each RSN pre-authentication session. If global library deinitialization is needed (i.e., one that is shared between both authentication types), the TLS library wrapper should maintain a reference counter and do global deinitialization only when moving from 1 to 0 references.

Definition at line 761 of file tls_openssl.c.

6.136.2.26 int tls_get_cipher (void ∗ *tls_ctx*, struct tls_connection ∗ *conn*, char ∗ *buf*, size_t *buflen*)

Get current cipher name.

Parameters:

tls_ctx TLS context data from tls_init()

conn Connection context data from tls_connection_init()

buf Buffer for the cipher name

buflen buf size

Returns:

0 on success, -1 on failure

Get the name of the currently used cipher.

Definition at line 2104 of file tls_openssl.c.

6.136.2.27 int tls_get_errors (void ∗ *tls_ctx*)

Process pending errors.

Parameters:

tls_ctx TLS context data from tls_init()

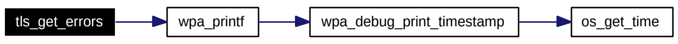
Returns:

Number of found error, 0 if no errors detected.

Process all pending TLS errors.

Definition at line 860 of file tls_openssl.c.

Here is the call graph for this function:



6.136.2.28 int tls_global_set_params (void ∗ *tls_ctx*, const struct tls_connection_params ∗ *params*)

Set TLS parameters for all TLS connection.

Parameters:

tls_ctx TLS context data from tls_init()

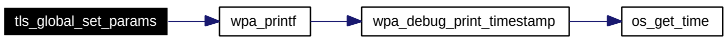
params Global TLS parameters

Returns:

0 on success, -1 on failure, TLS_SET_PARAMS_ENGINE_PRV_INIT_FAILED (-2) on possible PIN error causing PKCS#11 engine failure, or TLS_SET_PARAMS_ENGINE_PRV_VERIFY_FAILED (-3) on failure to verify the PKCS#11 engine private key.

Definition at line 2232 of file tls_openssl.c.

Here is the call graph for this function:



6.136.2.29 int tls_global_set_verify (void ∗ *tls_ctx*, int *check_crl*)

Set global certificate verification options.

Parameters:

tls_ctx TLS context data from tls_init()

check_crl 0 = do not verify CRLs, 1 = verify CRL for the user certificate, 2 = verify CRL for all certificates

Returns:

0 on success, -1 on failure

Definition at line 1193 of file tls_openssl.c.

6.136.2.30 void∗ tls_init (const struct tls_config ∗ *conf*)

Initialize TLS library.

Parameters:

conf Configuration data for TLS library

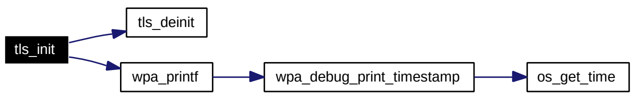
Returns:

Context data to be used as tls_ctx in calls to other functions, or NULL on failure.

Called once during program startup and once for each RSN pre-authentication session. In other words, there can be two concurrent TLS contexts. If global library initialization is needed (i.e., one that is shared between both authentication types), the TLS library wrapper should maintain a reference counter and do global initialization only when moving from 0 to 1 reference.

Definition at line 717 of file tls_openssl.c.

Here is the call graph for this function:



6.137 vlan_init.c File Reference

hostapd / VLAN initialization

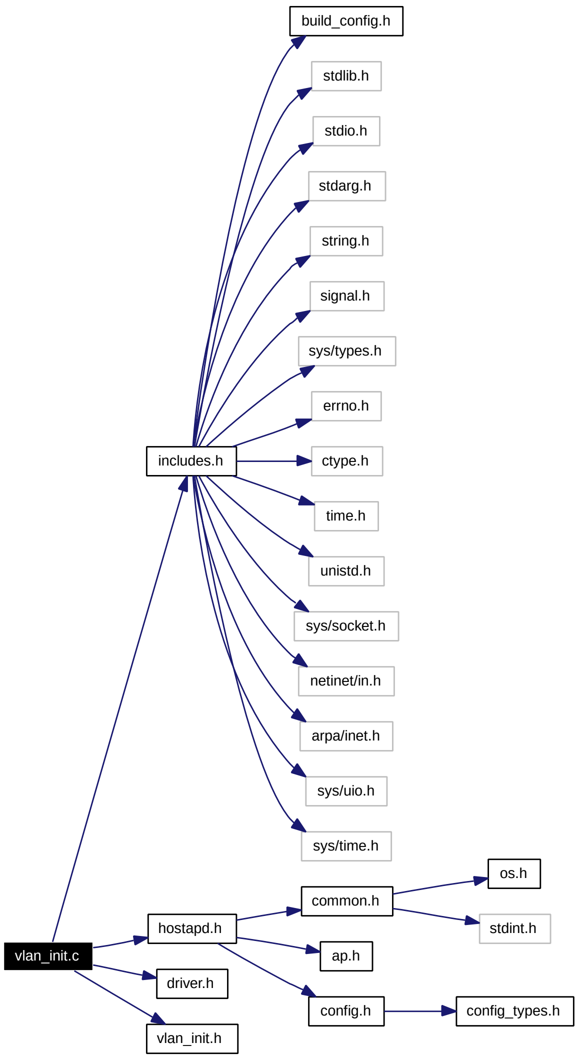
```
#include "includes.h"
```

```
#include "hostapd.h"
```

```
#include "driver.h"
```

```
#include "vlan_init.h"
```

Include dependency graph for vlan_init.c:



Functions

- int **vlan_setup_encryption_dyn** (struct hostapd_data ∗hapd, struct hostapd_ssid ∗mssid, const char ∗dyn_vlan)
- int **vlan_init** (struct hostapd_data ∗hapd)
- void **vlan_deinit** (struct hostapd_data ∗hapd)

- int **vlan_reconfig** (struct hostapd_data ∗hapd, struct hostapd_config ∗oldconf, struct hostapd_bss_-config ∗oldbss)
- hostapd_vlan ∗ **vlan_add_dynamic** (struct hostapd_data ∗hapd, struct hostapd_vlan ∗vlan, int vlan_id)
- int **vlan_remove_dynamic** (struct hostapd_data ∗hapd, int vlan_id)

6.137.1 Detailed Description

hostapd / VLAN initialization

Copyright

Copyright 2003, Instant802 Networks, Inc. Copyright 2005-2006, Devicescape Software, Inc.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

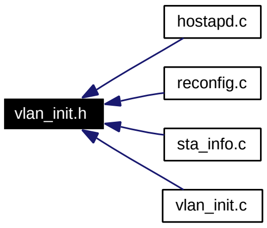
See README and COPYING for more details.

Definition in file vlan_init.c.

6.138 vlan_init.h File Reference

hostapd / VLAN initialization

This graph shows which files directly or indirectly include this file:



Functions

- int **vlan_init** (struct hostapd_data ∗hapd)
- void **vlan_deinit** (struct hostapd_data ∗hapd)
- int **vlan_reconfig** (struct hostapd_data ∗hapd, struct hostapd_config ∗oldconf, struct hostapd_bss_-config ∗oldbss)
- hostapd_vlan ∗ **vlan_add_dynamic** (struct hostapd_data ∗hapd, struct hostapd_vlan ∗vlan, int vlan_id)
- int **vlan_remove_dynamic** (struct hostapd_data ∗hapd, int vlan_id)
- int **vlan_setup_encryption_dyn** (struct hostapd_data ∗hapd, struct hostapd_ssid ∗mssid, const char ∗dyn_vlan)

6.138.1 Detailed Description

hostapd / VLAN initialization

Copyright

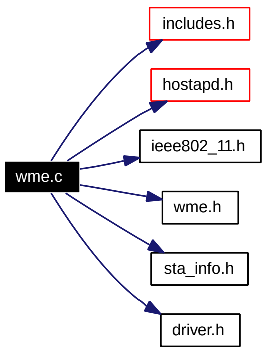
Definition in file vlan_init.h.

6.139 wme.c File Reference

hostapd / WMM (Wi-Fi Multimedia)

```
#include "includes.h"
#include "hostapd.h"
#include "ieee802_11.h"
#include "wme.h"
#include "sta_info.h"
#include "driver.h"
```

Include dependency graph for wme.c:



Functions

- u8 ∗ **hostapd_eid_wme** (struct hostapd_data ∗hapd, u8 ∗eid)
- int **hostapd_eid_wme_valid** (struct hostapd_data ∗hapd, u8 ∗eid, size_t len)
- int **hostapd_wme_sta_config** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **hostapd_wme_action** (struct hostapd_data ∗hapd, struct ieee80211_mgmt ∗mgmt, size_t len)

6.139.1 Detailed Description

hostapd / WMM (Wi-Fi Multimedia)

Copyright

Copyright 2002-2003, Instant802 Networks, Inc. Copyright 2005-2006, Devicescape Software, Inc.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

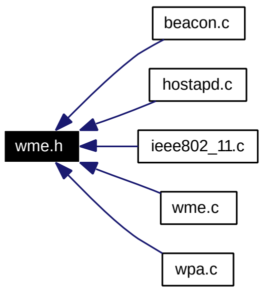
See README and COPYING for more details.

Definition in file wme.c.

6.140 wme.h File Reference

hostapd / WMM (Wi-Fi Multimedia)

This graph shows which files directly or indirectly include this file:



Defines

- #define **WME_OUI_TYPE** 2
- #define **WME_OUI_SUBTYPE_INFORMATION_ELEMENT** 0
- #define **WME_OUI_SUBTYPE_PARAMETER_ELEMENT** 1
- #define **WME_OUI_SUBTYPE_TSPEC_ELEMENT** 2
- #define **WME_VERSION** 1
- #define **WME_ACTION_CATEGORY** 17
- #define **WME_ACTION_CODE_SETUP_REQUEST** 0
- #define **WME_ACTION_CODE_SETUP_RESPONSE** 1
- #define **WME_ACTION_CODE_TEARDOWN** 2
- #define **WME_SETUP_RESPONSE_STATUS_ADMISSION_ACCEPTED** 0
- #define **WME_SETUP_RESPONSE_STATUS_INVALID_PARAMETERS** 1
- #define **WME_SETUP_RESPONSE_STATUS_REFUSED** 3
- #define **WME_TSPEC_DIRECTION_UPLINK** 0
- #define **WME_TSPEC_DIRECTION_DOWNLINK** 1
- #define **WME_TSPEC_DIRECTION_BI_DIRECTIONAL** 3

Enumerations

- enum { **WME_AC_BK** = 1, **WME_AC_BE** = 0, **WME_AC_VI** = 2, **WME_AC_VO** = 3 }

Functions

- u16 **tsinfo** (int tag1d, int contention_based, int direction)
- u8 ∗ **hostapd_eid_wme** (struct hostapd_data ∗hapd, u8 ∗eid)
- int **hostapd_eid_wme_valid** (struct hostapd_data ∗hapd, u8 ∗eid, size_t len)
- int **hostapd_wme_sta_config** (struct hostapd_data ∗hapd, struct sta_info ∗sta)
- void **hostapd_wme_action** (struct hostapd_data ∗hapd, struct ieee80211_mgmt ∗mgmt, size_t len)

Variables

- wme_information_element **packed**

6.140.1 Detailed Description

hostapd / WMM (Wi-Fi Multimedia)

Copyright

Copyright 2002-2003, Instant802 Networks, Inc. Copyright 2005-2006, Devicescape Software, Inc.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

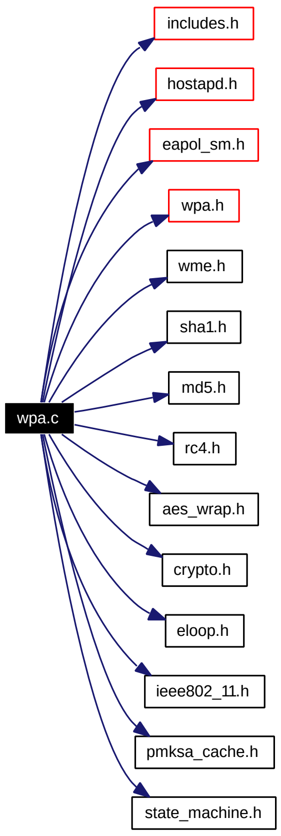
Definition in file wme.h.

6.141 wpa.c File Reference

hostapd - IEEE 802.11i-2004 / WPA Authenticator

```
#include "includes.h"
#include "hostapd.h"
#include "eapol_sm.h"
#include "wpa.h"
#include "wme.h"
#include "sha1.h"
#include "md5.h"
#include "rc4.h"
#include "aes_wrap.h"
#include "crypto.h"
#include "eloop.h"
#include "ieee802_11.h"
#include "pmksa_cache.h"
#include "state_machine.h"
```

Include dependency graph for wpa.c:



Defines

- #define **STATE_MACHINE_DATA** struct wpa_state_machine
- #define **STATE_MACHINE_DEBUG_PREFIX** "WPA"
- #define **STATE_MACHINE_ADDR** sm → addr
- #define **RSN_NUM_REPLAY_COUNTERS_1** 0
- #define **RSN_NUM_REPLAY_COUNTERS_2** 1
- #define **RSN_NUM_REPLAY_COUNTERS_4** 2
- #define **RSN_NUM_REPLAY_COUNTERS_16** 3
- #define **GENERIC_INFO_ELEM** 0xdd
- #define **RSN_INFO_ELEM** 0x30
- #define **RSN_SUITE** "%02x-%02x-%02x-%d"
- #define **RSN_SUITE_ARG**(s) (s)[0], (s)[1], (s)[2], (s)[3]

Functions

- wpa_authenticator ∗ wpa_init (const u8 ∗addr, struct wpa_auth_config ∗conf, struct wpa_auth_-callbacks ∗cb)

  Initialize WPA authenticator.

- void wpa_deinit (struct wpa_authenticator ∗wpa_auth)

  Deinitialize WPA authenticator.

- int wpa_reconfig (struct wpa_authenticator ∗wpa_auth, struct wpa_auth_config ∗conf)

  Update WPA authenticator configuration.

- int **wpa_validate_wpa_ie** (struct wpa_authenticator ∗wpa_auth, struct wpa_state_machine ∗sm, const u8 ∗wpa_ie, size_t wpa_ie_len)
- wpa_state_machine ∗ **wpa_auth_sta_init** (struct wpa_authenticator ∗wpa_auth, const u8 ∗addr)
- void **wpa_auth_sta_associated** (struct wpa_authenticator ∗wpa_auth, struct wpa_state_machine ∗sm)
- void **wpa_auth_sta_deinit** (struct wpa_state_machine ∗sm)
- void **wpa_receive** (struct wpa_authenticator ∗wpa_auth, struct wpa_state_machine ∗sm, u8 ∗data, size_t data_len)
- void **wpa_remove_ptk** (struct wpa_state_machine ∗sm)
- void **wpa_auth_sm_event** (struct wpa_state_machine ∗sm, wpa_event event)
- **SM_STATE** (WPA_PTK, INITIALIZE)
- **SM_STATE** (WPA_PTK, DISCONNECT)
- **SM_STATE** (WPA_PTK, AUTHENTICATION)
- **SM_STATE** (WPA_PTK, INITPMK)
- **SM_STATE** (WPA_PTK, INITPSK)
- **SM_STATE** (WPA_PTK, PTKSTART)
- **SM_STATE** (WPA_PTK, PTKCALCNEGOTIATING)
- **SM_STATE** (WPA_PTK, PTKINITNEGOTIATING)
- **SM_STATE** (WPA_PTK, PTKINITDONE)
- **SM_STEP** (WPA_PTK)
- **SM_STATE** (WPA_PTK_GROUP, IDLE)
- **SM_STATE** (WPA_PTK_GROUP, REKEYNEGOTIATING)
- **SM_STATE** (WPA_PTK_GROUP, REKEYESTABLISHED)
- **SM_STATE** (WPA_PTK_GROUP, KEYERROR)

- void **wpa_auth_sm_notify** (struct wpa_state_machine ∗sm)
- void **wpa_gtk_rekey** (struct wpa_authenticator ∗wpa_auth)
- int **wpa_get_mib** (struct wpa_authenticator ∗wpa_auth, char ∗buf, size_t buflen)
- int **wpa_get_mib_sta** (struct wpa_state_machine ∗sm, char ∗buf, size_t buflen)
- void **wpa_auth_countermeasures_start** (struct wpa_authenticator ∗wpa_auth)
- int **wpa_auth_pairwise_set** (struct wpa_state_machine ∗sm)
- int **wpa_auth_sta_key_mgmt** (struct wpa_state_machine ∗sm)
- int **wpa_auth_sta_wpa_version** (struct wpa_state_machine ∗sm)
- int **wpa_auth_sta_clear_pmksa** (struct wpa_state_machine ∗sm, struct rsn_pmksa_cache_entry ∗entry)
- rsn_pmksa_cache_entry ∗ **wpa_auth_sta_get_pmksa** (struct wpa_state_machine ∗sm)
- void **wpa_auth_sta_local_mic_failure_report** (struct wpa_state_machine ∗sm)
- const u8 ∗ **wpa_auth_get_wpa_ie** (struct wpa_authenticator ∗wpa_auth, size_t ∗len)
- int **wpa_auth_pmksa_add** (struct wpa_state_machine ∗sm, const u8 ∗pmk, int session_timeout, struct eapol_state_machine ∗eapol)
- int **wpa_auth_pmksa_add_preauth** (struct wpa_authenticator ∗wpa_auth, const u8 ∗pmk, size_t len, const u8 ∗sta_addr, int session_timeout, struct eapol_state_machine ∗eapol)
- int **wpa_auth_sta_set_vlan** (struct wpa_state_machine ∗sm, int vlan_id)

Variables

- wpa_ie_hdr **STRUCT_PACKED**

6.141.1 Detailed Description

hostapd - IEEE 802.11i-2004 / WPA Authenticator

Copyright

Definition in file wpa.c.

6.141.2 Function Documentation

6.141.2.1 void wpa_deinit (struct wpa_authenticator ∗ *wpa_auth*)

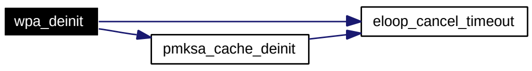
Deinitialize WPA authenticator.

Parameters:

wpa_auth Pointer to WPA authenticator data from wpa_init()

Definition at line 908 of file wpa.c.

Here is the call graph for this function:



6.141.2.2 struct wpa_authenticator∗ wpa_init (const u8 ∗ *addr*, struct wpa_auth_config ∗ *conf*, struct wpa_auth_callbacks ∗ *cb*)

Initialize WPA authenticator.

Parameters:

addr Authenticator address

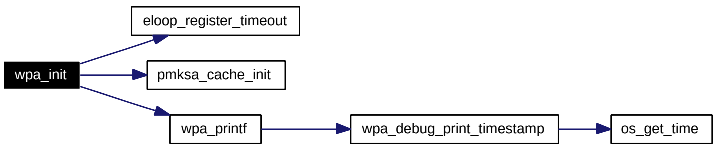
conf Configuration for WPA authenticator

Returns:

Pointer to WPA authenticator data or NULL on failure

Definition at line 855 of file wpa.c.

Here is the call graph for this function:



6.141.2.3 int wpa_reconfig (struct wpa_authenticator ∗ *wpa_auth*, struct wpa_auth_config ∗ *conf*)

Update WPA authenticator configuration.

Parameters:

wpa_auth Pointer to WPA authenticator data from wpa_init()

conf Configuration for WPA authenticator

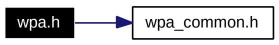
Definition at line 939 of file wpa.c.

6.142 wpa.h File Reference

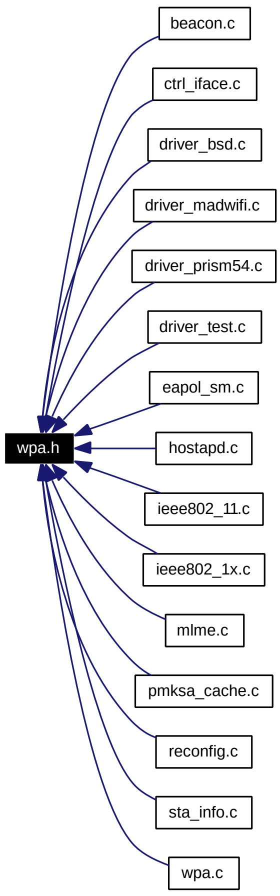
hostapd - IEEE 802.11i-2004 / WPA Authenticator

`#include "wpa_common.h"`

Include dependency graph for wpa.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **WPA_PMK_LEN** PMK_LEN
- #define **WPA_GMK_LEN** 32
- #define **WPA_GTK_MAX_LEN** 32
- #define **PMKID_LEN** 16
- #define **WPA_CAPABILITY_PREAUTH** BIT(0)
- #define **WPA_CAPABILITY_MGMT_FRAME_PROTECTION** BIT(6)
- #define **WPA_CAPABILITY_PEERKEY_ENABLED** BIT(9)

- #define **WPA_KEY_INFO_TYPE_MASK** (BIT(0) | BIT(1) | BIT(2))
- #define **WPA_KEY_INFO_TYPE_HMAC_MD5_RC4** BIT(0)
- #define **WPA_KEY_INFO_TYPE_HMAC_SHA1_AES** BIT(1)
- #define **WPA_KEY_INFO_KEY_TYPE** BIT(3)
- #define **WPA_KEY_INFO_KEY_INDEX_MASK** (BIT(4) | BIT(5))
- #define **WPA_KEY_INFO_KEY_INDEX_SHIFT** 4
- #define **WPA_KEY_INFO_INSTALL** BIT(6)
- #define **WPA_KEY_INFO_TXRX** BIT(6)
- #define **WPA_KEY_INFO_ACK** BIT(7)
- #define **WPA_KEY_INFO_MIC** BIT(8)
- #define **WPA_KEY_INFO_SECURE** BIT(9)
- #define **WPA_KEY_INFO_ERROR** BIT(10)
- #define **WPA_KEY_INFO_REQUEST** BIT(11)
- #define **WPA_KEY_INFO_ENCR_KEY_DATA** BIT(12)
- #define **WPA_KEY_INFO_SMK_MESSAGE** BIT(13)

Enumerations

- enum **logger_level** { **LOGGER_DEBUG**, **LOGGER_INFO**, **LOGGER_WARNING** }
- enum **wpa_eapol_variable** {

  WPA_EAPOL_portEnabled, **WPA_EAPOL_portValid**, **WPA_EAPOL_authorized**, **WPA_-EAPOL_portControl_Auto**,

  WPA_EAPOL_keyRun, **WPA_EAPOL_keyAvailable**, **WPA_EAPOL_keyDone**, **WPA_-EAPOL_inc_EapolFramesTx** }
- enum {

  WPA_IE_OK, **WPA_INVALID_IE**, **WPA_INVALID_GROUP**, **WPA_INVALID_PAIRWISE**,

  WPA_INVALID_AKMP, **WPA_NOT_ENABLED**, **WPA_ALLOC_FAIL**, **WPA_MGMT_-FRAME_PROTECTION_VIOLATION**,

  WPA_INVALID_MGMT_GROUP_CIPHER }
- enum **wpa_event** {

  WPA_AUTH, **WPA_ASSOC**, **WPA_DISASSOC**, **WPA_DEAUTH**,

  WPA_REAUTH, **WPA_REAUTH_EAPOL** }

Functions

- wpa_authenticator ∗ wpa_init (const u8 ∗addr, struct wpa_auth_config ∗conf, struct wpa_auth_-callbacks ∗cb)

  Initialize WPA authenticator.

- void wpa_deinit (struct wpa_authenticator ∗wpa_auth)

  Deinitialize WPA authenticator.

- int wpa_reconfig (struct wpa_authenticator ∗wpa_auth, struct wpa_auth_config ∗conf)

  Update WPA authenticator configuration.

- int **wpa_validate_wpa_ie** (struct wpa_authenticator ∗wpa_auth, struct wpa_state_machine ∗sm, const u8 ∗wpa_ie, size_t wpa_ie_len)
- wpa_state_machine ∗ **wpa_auth_sta_init** (struct wpa_authenticator ∗wpa_auth, const u8 ∗addr)

- void **wpa_auth_sta_associated** (struct wpa_authenticator ∗wpa_auth, struct wpa_state_machine ∗sm)
- void **wpa_auth_sta_deinit** (struct wpa_state_machine ∗sm)
- void **wpa_receive** (struct wpa_authenticator ∗wpa_auth, struct wpa_state_machine ∗sm, u8 ∗data, size_t data_len)
- void **wpa_remove_ptk** (struct wpa_state_machine ∗sm)
- void **wpa_auth_sm_event** (struct wpa_state_machine ∗sm, wpa_event event)
- void **wpa_auth_sm_notify** (struct wpa_state_machine ∗sm)
- void **wpa_gtk_rekey** (struct wpa_authenticator ∗wpa_auth)
- int **wpa_get_mib** (struct wpa_authenticator ∗wpa_auth, char ∗buf, size_t buflen)
- int **wpa_get_mib_sta** (struct wpa_state_machine ∗sm, char ∗buf, size_t buflen)
- void **wpa_auth_countermeasures_start** (struct wpa_authenticator ∗wpa_auth)
- int **wpa_auth_pairwise_set** (struct wpa_state_machine ∗sm)
- int **wpa_auth_sta_key_mgmt** (struct wpa_state_machine ∗sm)
- int **wpa_auth_sta_wpa_version** (struct wpa_state_machine ∗sm)
- int **wpa_auth_sta_clear_pmksa** (struct wpa_state_machine ∗sm, struct rsn_pmksa_cache_entry ∗entry)
- rsn_pmksa_cache_entry ∗ **wpa_auth_sta_get_pmksa** (struct wpa_state_machine ∗sm)
- void **wpa_auth_sta_local_mic_failure_report** (struct wpa_state_machine ∗sm)
- const u8 ∗ **wpa_auth_get_wpa_ie** (struct wpa_authenticator ∗wpa_auth, size_t ∗len)
- int **wpa_auth_pmksa_add** (struct wpa_state_machine ∗sm, const u8 ∗pmk, int session_timeout, struct eapol_state_machine ∗eapol)
- int **wpa_auth_pmksa_add_preauth** (struct wpa_authenticator ∗wpa_auth, const u8 ∗pmk, size_t len, const u8 ∗sta_addr, int session_timeout, struct eapol_state_machine ∗eapol)
- int **wpa_auth_sta_set_vlan** (struct wpa_state_machine ∗sm, int vlan_id)

Variables

- wpa_eapol_key **packed**

6.142.1 Detailed Description

hostapd - IEEE 802.11i-2004 / WPA Authenticator

Copyright

Copyright (c) 2004-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file wpa.h.

6.142.2 Function Documentation

6.142.2.1 void wpa_deinit (struct wpa_authenticator ∗ *wpa_auth*)

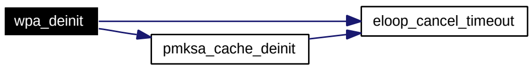
Deinitialize WPA authenticator.

Parameters:

wpa_auth Pointer to WPA authenticator data from wpa_init()

Definition at line 908 of file wpa.c.

Here is the call graph for this function:



6.142.2.2 struct wpa_authenticator∗ wpa_init (const u8 ∗ *addr*, struct wpa_auth_config ∗ *conf*, struct wpa_auth_callbacks ∗ *cb*)

Initialize WPA authenticator.

Parameters:

addr Authenticator address

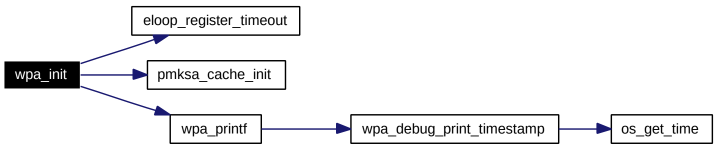
conf Configuration for WPA authenticator

Returns:

Pointer to WPA authenticator data or NULL on failure

Definition at line 855 of file wpa.c.

Here is the call graph for this function:



6.142.2.3 int wpa_reconfig (struct wpa_authenticator ∗ *wpa_auth*, struct wpa_auth_config ∗ *conf*)

Update WPA authenticator configuration.

Parameters:

wpa_auth Pointer to WPA authenticator data from wpa_init()

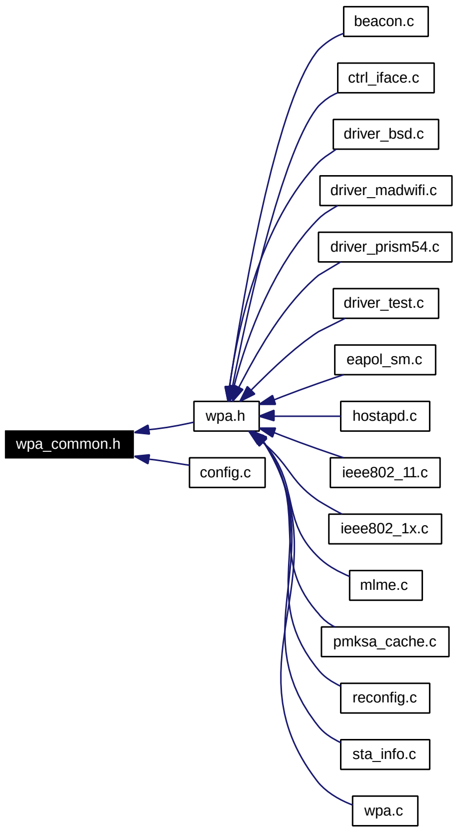
conf Configuration for WPA authenticator

Definition at line 939 of file wpa.c.

6.143 wpa_common.h File Reference

WPA definitions shared between hostapd and wpa_supplicant.

This graph shows which files directly or indirectly include this file:



Defines

- #define **WPA_REPLAY_COUNTER_LEN** 8
- #define **WPA_NONCE_LEN** 32
- #define **WPA_KEY_RSC_LEN** 8
- #define **EAPOL_VERSION** 2

Enumerations

- enum {

  IEEE802_1X_TYPE_EAP_PACKET = 0, **IEEE802_1X_TYPE_EAPOL_START** = 1, **IEEE802_1X_TYPE_EAPOL_LOGOFF** = 2, **IEEE802_1X_TYPE_EAPOL_KEY** = 3,

  IEEE802_1X_TYPE_EAPOL_ENCAPSULATED_ASF_ALERT = 4 }
- enum { **EAPOL_KEY_TYPE_RC4** = 1, **EAPOL_KEY_TYPE_RSN** = 2, **EAPOL_KEY_-TYPE_WPA** = 254 }

Variables

- ieee802_1x_hdr **STRUCT_PACKED**

6.143.1 Detailed Description

WPA definitions shared between hostapd and wpa_supplicant.

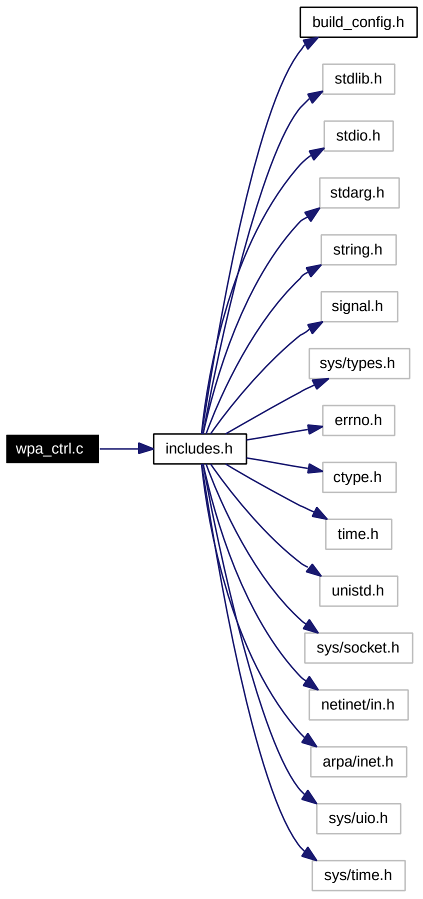
Definition in file wpa_common.h.

6.144 wpa_ctrl.c File Reference

wpa_supplicant/hostapd control interface library

```
#include "includes.h"
```

Include dependency graph for wpa_ctrl.c:



6.144.1 Detailed Description

wpa_supplicant/hostapd control interface library

Copyright

Copyright (c) 2004-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

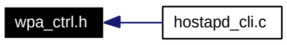
See README and COPYING for more details.

Definition in file wpa_ctrl.c.

6.145 wpa_ctrl.h File Reference

wpa_supplicant/hostapd control interface library

This graph shows which files directly or indirectly include this file:



Defines

- #define WPA_CTRL_REQ "CTRL-REQ-"
- #define WPA_CTRL_RSP "CTRL-RSP-"
- #define WPA_EVENT_CONNECTED "CTRL-EVENT-CONNECTED "
- #define WPA_EVENT_DISCONNECTED "CTRL-EVENT-DISCONNECTED "
- #define WPA_EVENT_TERMINATING "CTRL-EVENT-TERMINATING "
- #define WPA_EVENT_PASSWORD_CHANGED "CTRL-EVENT-PASSWORD-CHANGED "
- #define WPA_EVENT_EAP_NOTIFICATION "CTRL-EVENT-EAP-NOTIFICATION "
- #define WPA_EVENT_EAP_STARTED "CTRL-EVENT-EAP-STARTED "
- #define WPA_EVENT_EAP_METHOD "CTRL-EVENT-EAP-METHOD "
- #define WPA_EVENT_EAP_SUCCESS "CTRL-EVENT-EAP-SUCCESS "
- #define WPA_EVENT_EAP_FAILURE "CTRL-EVENT-EAP-FAILURE "

Functions

- wpa_ctrl ∗ wpa_ctrl_open (const char ∗ctrl_path)

  Open a control interface to wpa_supplicant/hostapd.

- void wpa_ctrl_close (struct wpa_ctrl ∗ctrl)

  Close a control interface to wpa_supplicant/hostapd.

- int wpa_ctrl_request (struct wpa_ctrl ∗ctrl, const char ∗cmd, size_t cmd_len, char ∗reply, size_t ∗reply_len, void(∗msg_cb)(char ∗msg, size_t len))

  Send a command to wpa_supplicant/hostapd.

- int wpa_ctrl_attach (struct wpa_ctrl ∗ctrl)

  Register as an event monitor for the control interface.

- int wpa_ctrl_detach (struct wpa_ctrl ∗ctrl)

  Unregister event monitor from the control interface.

- int wpa_ctrl_recv (struct wpa_ctrl ∗ctrl, char ∗reply, size_t ∗reply_len)

  Receive a pending control interface message.

- int wpa_ctrl_pending (struct wpa_ctrl ∗ctrl)

  Check whether there are pending event messages.

- int wpa_ctrl_get_fd (struct wpa_ctrl ∗ctrl)

  Get file descriptor used by the control interface.

6.145.1 Detailed Description

wpa_supplicant/hostapd control interface library

Copyright

Copyright (c) 2004-2006, Jouni Malinen <`jkmaline@cc.hut.fi`>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

Alternatively, this software may be distributed under the terms of BSD license.

See README and COPYING for more details.

Definition in file wpa_ctrl.h.

6.145.2 Define Documentation

6.145.2.1 #define WPA_CTRL_REQ "CTRL-REQ-"

Interactive request for identity/password/pin

Definition at line 26 of file wpa_ctrl.h.

6.145.2.2 #define WPA_CTRL_RSP "CTRL-RSP-"

Response to identity/password/pin request

Definition at line 29 of file wpa_ctrl.h.

6.145.2.3 #define WPA_EVENT_CONNECTED "CTRL-EVENT-CONNECTED "

Authentication completed successfully and data connection enabled

Definition at line 33 of file wpa_ctrl.h.

6.145.2.4 #define WPA_EVENT_DISCONNECTED "CTRL-EVENT-DISCONNECTED "

Disconnected, data connection is not available

Definition at line 35 of file wpa_ctrl.h.

6.145.2.5 #define WPA_EVENT_EAP_FAILURE "CTRL-EVENT-EAP-FAILURE "

EAP authentication failed (EAP-Failure received)

Definition at line 49 of file wpa_ctrl.h.

6.145.2.6 #define WPA_EVENT_EAP_METHOD "CTRL-EVENT-EAP-METHOD "

EAP method selected

Definition at line 45 of file wpa_ctrl.h.

6.145.2.7 #define WPA_EVENT_EAP_NOTIFICATION "CTRL-EVENT-EAP-NOTIFICATION "

EAP-Request/Notification received

Definition at line 41 of file wpa_ctrl.h.

6.145.2.8 #define WPA_EVENT_EAP_STARTED "CTRL-EVENT-EAP-STARTED "

EAP authentication started (EAP-Request/Identity received)

Definition at line 43 of file wpa_ctrl.h.

6.145.2.9 #define WPA_EVENT_EAP_SUCCESS "CTRL-EVENT-EAP-SUCCESS "

EAP authentication completed successfully

Definition at line 47 of file wpa_ctrl.h.

6.145.2.10 #define WPA_EVENT_PASSWORD_CHANGED "CTRL-EVENT-PASSWORD-CHANGED "

Password change was completed successfully

Definition at line 39 of file wpa_ctrl.h.

6.145.2.11 #define WPA_EVENT_TERMINATING "CTRL-EVENT-TERMINATING "

wpa_supplicant is exiting

Definition at line 37 of file wpa_ctrl.h.

6.145.3 Function Documentation

6.145.3.1 int wpa_ctrl_attach (struct wpa_ctrl ∗ *ctrl*)

Register as an event monitor for the control interface.

Parameters:

ctrl Control interface data from wpa_ctrl_open()

Returns:

0 on success, -1 on failure, -2 on timeout

This function registers the control interface connection as a monitor for wpa_supplicant/hostapd events. After a success wpa_ctrl_attach() call, the control interface connection starts receiving event messages that can be read with wpa_ctrl_recv().

6.145.3.2 void wpa_ctrl_close (struct wpa_ctrl ∗ *ctrl*)

Close a control interface to wpa_supplicant/hostapd.

Parameters:

ctrl Control interface data from wpa_ctrl_open()

This function is used to close a control interface.

6.145.3.3 int wpa_ctrl_detach (struct wpa_ctrl ∗ *ctrl*)

Unregister event monitor from the control interface.

Parameters:

ctrl Control interface data from wpa_ctrl_open()

Returns:

0 on success, -1 on failure, -2 on timeout

This function unregisters the control interface connection as a monitor for wpa_supplicant/hostapd events, i.e., cancels the registration done with wpa_ctrl_attach().

6.145.3.4 int wpa_ctrl_get_fd (struct wpa_ctrl ∗ *ctrl*)

Get file descriptor used by the control interface.

Parameters:

ctrl Control interface data from wpa_ctrl_open()

Returns:

File descriptor used for the connection

This function can be used to get the file descriptor that is used for the control interface connection. The returned value can be used, e.g., with select() while waiting for multiple events.

The returned file descriptor must not be used directly for sending or receiving packets; instead, the library functions wpa_ctrl_request() and wpa_ctrl_recv() must be used for this.

6.145.3.5 struct wpa_ctrl∗ wpa_ctrl_open (const char ∗ *ctrl_path*)

Open a control interface to wpa_supplicant/hostapd.

Parameters:

ctrl_path Path for UNIX domain sockets; ignored if UDP sockets are used.

Returns:

Pointer to abstract control interface data or NULL on failure

This function is used to open a control interface to wpa_supplicant/hostapd. ctrl_path is usually /var/run/wpa_supplicant or /var/run/hostapd. This path is configured in wpa_supplicant/hostapd and other programs using the control interface need to use matching path configuration.

6.145.3.6 int wpa_ctrl_pending (struct wpa_ctrl ∗ *ctrl*)

Check whether there are pending event messages.

Parameters:

ctrl Control interface data from wpa_ctrl_open()

Returns:

1 if there are pending messages, 0 if no, or -1 on error

This function will check whether there are any pending control interface message available to be received with wpa_ctrl_recv(). wpa_ctrl_pending() is only used for event messages, i.e., wpa_ctrl_attach() must have been used to register the control interface as an event monitor.

6.145.3.7 int wpa_ctrl_recv (struct wpa_ctrl ∗ *ctrl*, char ∗ *reply*, size_t ∗ *reply_len*)

Receive a pending control interface message.

Parameters:

ctrl Control interface data from wpa_ctrl_open()

reply Buffer for the message data

reply_len Length of the reply buffer

Returns:

0 on success, -1 on failure

This function will receive a pending control interface message. This function will block if no messages are available. The received response will be written to reply and reply_len is set to the actual length of the reply. wpa_ctrl_recv() is only used for event messages, i.e., wpa_ctrl_attach() must have been used to register the control interface as an event monitor.

6.145.3.8 int wpa_ctrl_request (struct wpa_ctrl ∗ *ctrl*, const char ∗ *cmd*, size_t *cmd_len*, char ∗ *reply*, size_t ∗ *reply_len*, void(∗)(char ∗msg, size_t len) *msg_cb*)

Send a command to wpa_supplicant/hostapd.

Parameters:

ctrl Control interface data from wpa_ctrl_open()

cmd Command; usually, ASCII text, e.g., "PING"

cmd_len Length of the cmd in bytes

reply Buffer for the response

reply_len Reply buffer length

msg_cb Callback function for unsolicited messages or NULL if not used

Returns:

0 on success, -1 on error (send or receive failed), -2 on timeout

This function is used to send commands to wpa_supplicant/hostapd. Received response will be written to reply and reply_len is set to the actual length of the reply. This function will block for up to two seconds while waiting for the reply. If unsolicited messages are received, the blocking time may be longer.

msg_cb can be used to register a callback function that will be called for unsolicited messages received while waiting for the command response. These messages may be received if wpa_ctrl_request() is called at the same time as wpa_supplicant/hostapd is sending such a message. This can happen only if the program has used wpa_ctrl_attach() to register itself as a monitor for event messages. Alternatively to msg_cb, programs can register two control interface connections and use one of them for commands and the other one for receiving event messages, in other words, call wpa_ctrl_attach() only for the control interface connection that will be used for event messages.

Chapter 7

hostapd Page Documentation

7.1 Structure of the source code

7.2 Control interface

hostapd implements a control interface that can be used by external programs to control the operations of the hostapd daemon and to get status information and event notifications. There is a small C library, in a form of a single C file, wpa_ctrl.c, that provides helper functions to facilitate the use of the control interface. External programs can link this file into them and then use the library functions documented in wpa_ctrl.h to interact with wpa_supplicant. This library can also be used with C++. hostapd_cli.c is an example program using this library.

There are multiple mechanisms for inter-process communication. For example, Linux version of hostapd is using UNIX domain sockets for the control interface. The use of the functions defined in wpa_ctrl.h can be used to hide the details of the used IPC from external programs.

7.2.1 Using the control interface

External programs, e.g., a GUI or a configuration utility, that need to communicate with hostapd should link in wpa_ctrl.c. This allows them to use helper functions to open connection to the control interface with wpa_ctrl_open() and to send commands with wpa_ctrl_request().

hostapd uses the control interface for two types of communication: commands and unsolicited event messages. Commands are a pair of messages, a request from the external program and a response from hostapd. These can be executed using wpa_ctrl_request(). Unsolicited event messages are sent by hostapd to the control interface connection without specific request from the external program for receiving each message. However, the external program needs to attach to the control interface with wpa_ctrl_attach() to receive these unsolicited messages.

If the control interface connection is used both for commands and unsolicited event messages, there is potential for receiving an unsolicited message between the command request and response. wpa_ctrl_-request() caller will need to supply a callback, msg_cb, for processing these messages. Often it is easier to open two control interface connections by calling wpa_ctrl_open() twice and then use one of the connections for commands and the other one for unsolicited messages. This way command request/response pairs will not be broken by unsolicited messages. wpa_cli is an example of how to use only one connection for both purposes and wpa_gui demonstrates how to use two separate connections.

Once the control interface connection is not needed anymore, it should be closed by calling wpa_ctrl_-close(). If the connection was used for unsolicited event messages, it should be first detached by calling wpa_ctrl_detach().

7.2.2 Control interface commands

Following commands can be used with wpa_ctrl_request():

7.2.2.1 PING

This command can be used to test whether hostapd is replying to the control interface commands. The expected reply is `PONG` if the connection is open and hostapd is processing commands.

7.3 Driver wrapper implementation (driver.h, drivers.c)

All hardware and driver dependent functionality is in separate C files that implement defined wrapper functions. Other parts of the hostapd are designed to be hardware, driver, and operating system independent.

Driver wrappers need to implement whatever calls are used in the target operating system/driver for controlling wireless LAN devices. As an example, in case of Linux, these are mostly some glue code and ioctl() calls and netlink message parsing for Linux Wireless Extensions (WE). Since features required for WPA were added only recently to Linux Wireless Extensions (in version 18), some driver specific code is used in number of driver interface implementations. These driver dependent parts can be replaced with generic code in driver_wext.c once the target driver includes full support for WE-18. After that, all Linux drivers, at least in theory, could use the same driver wrapper code.

7.4 EAP server implementation

Extensible Authentication Protocol (EAP) is an authentication framework defined in RFC 3748. hostapd uses a separate code module for EAP server implementation. This module was designed to use only a minimal set of direct function calls (mainly, to debug/event functions) in order for it to be usable in other programs. The design of the EAP implementation is based loosely on RFC 4137. The state machine is defined in this RFC and so is the interface between the server state machine and methods. As such, this RFC provides useful information for understanding the EAP server implementation in hostapd.

Some of the terminology used in EAP state machine is referring to EAPOL (IEEE 802.1X), but there is no strict requirement on the lower layer being IEEE 802.1X if EAP module is built for other programs than wpa_supplicant. These terms should be understood to refer to the lower layer as defined in RFC 4137.

7.4.1 Adding EAP methods

Each EAP method is implemented as a separate module, usually as one C file named eap_<name of the method>.c, e.g., eap_md5.c. All EAP methods use the same interface between the server state machine and method specific functions. This allows new EAP methods to be added without modifying the core EAP state machine implementation.

New EAP methods need to be registered by adding them into the build (Makefile) and the EAP method registration list in the eap_server_register_methods() function of eap_methods.c. Each EAP method should use a build-time configuration option, e.g., EAP_TLS, in order to make it possible to select which of the methods are included in the build.

EAP methods must implement the interface defined in eap_i.h. struct eap_method defines the needed function pointers that each EAP method must provide. In addition, the EAP type and name are registered using this structure. This interface is based on section 4.4 of RFC 4137.

It is recommended that the EAP methods would use generic helper functions, eap_msg_alloc() and eap_-hdr_validate() when processing messages. This allows code sharing and can avoid missing some of the needed validation steps for received packets. In addition, these functions make it easier to change between expanded and legacy EAP header, if needed.

When adding an EAP method that uses a vendor specific EAP type (Expanded Type as defined in RFC 3748, Chapter 5.7), the new method must be registered by passing vendor id instead of EAP_VENDOR_IETF to eap_server_method_alloc(). These methods must not try to emulate expanded types by registering a legacy EAP method for type 254. See eap_vendor_test.c for an example of an EAP method implementation that is implemented as an expanded type.

7.5 Porting to different target boards and operating systems

Index