Servidor Central - Control de Ascensores

2.0

Generated by Doxygen 1.9.8

Chapter 1

Data Structure Documentation

1.1 dtls_config_t Struct Reference

Configuración de seguridad DTLS-PSK.

`#include <dtls_common_config.h>`

Data Fields

- char * **psk_file**
- int **psk_timeout**
- int **dtls_mtu**
- int **retransmit_timeout**
- int **max_connections**
- int **session_cache_size**

1.1.1 Detailed Description

Configuración de seguridad DTLS-PSK.

Esta estructura contiene todos los parámetros necesarios para configurar la seguridad DTLS-PSK del servidor central.

1.1.2 Field Documentation

1.1.2.1 dtls_mtu

`int dtls_config_t::dtls_mtu`

MTU para DTLS en bytes

1.1.2.2 max_connections

```
int dtls_config_t::max_connections
```

Número máximo de conexiones simultáneas

1.1.2.3 psk_file

```
char* dtls_config_t::psk_file
```

Ruta al archivo de claves PSK

1.1.2.4 psk_timeout

```
int dtls_config_t::psk_timeout
```

Timeout de sesión PSK en segundos

1.1.2.5 retransmit_timeout

```
int dtls_config_t::retransmit_timeout
```

Timeout de retransmisión en segundos

1.1.2.6 session_cache_size

```
int dtls_config_t::session_cache_size
```

Tamaño del caché de sesiones

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

- include/servidor_central/ **dtls_common_config.h**

1.2 logging_config_t Struct Reference

Configuración del sistema de logging.

```
#include <logging.h>
```

Data Fields

- **log_level_t min_level**
- FILE * **log_file**
- int **use_timestamps**
- int **use_colors**
- int **use_thread_id**
- char * **log_format**

1.2.1 Detailed Description

Configuración del sistema de logging.

Esta estructura contiene la configuración del sistema de logging, incluyendo el nivel mínimo, el archivo de salida y las opciones de formato.

1.2.2 Field Documentation

1.2.2.1 log_file

```
FILE* logging_config_t::log_file
```

Archivo de salida para logs

1.2.2.2 log_format

```
char* logging_config_t::log_format
```

Formato personalizado de log

1.2.2.3 min_level

```
log_level_t logging_config_t::min_level
```

Nivel mínimo de logging

1.2.2.4 use_colors

```
int logging_config_t::use_colors
```

1 para colores en terminal, 0 para no

1.2.2.5 use_thread_id

```
int logging_config_t::use_thread_id
```

1 para incluir ID de thread, 0 para no

1.2.2.6 use_timestamps

```
int logging_config_t::use_timestamps
```

1 para incluir timestamps, 0 para no

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

- include/servidor_central/ **logging.h**

1.3 psk_valid_keys_t Struct Reference

Estructura para almacenar las claves PSK válidas.

Data Fields

- char ** **keys**
- int **count**
- int **capacity**

1.3.1 Detailed Description

Estructura para almacenar las claves PSK válidas.

Esta estructura mantiene en memoria todas las claves PSK válidas cargadas desde el archivo de claves. Utiliza un array dinámico para almacenar las claves como strings.

1.3.2 Field Documentation

1.3.2.1 capacity

```
int psk_valid_keys_t::capacity
```

Capacidad total del array

1.3.2.2 count

```
int psk_valid_keys_t::count
```

Número actual de claves cargadas

1.3.2.3 keys

```
char** psk_valid_keys_t::keys
```

Array de punteros a claves PSK

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

- src/ **psk_validator.c**

1.4 server_context_t Struct Reference

Estructura para manejar el contexto del servidor.

```
#include <server_functions.h>
```

Data Fields

- coap_context_t * **ctx**
- SSL_CTX * **ssl_ctx**
- sqlite3 * **db**
- int **running**
- char * **psk_file**
- int **port**

1.4.1 Detailed Description

Estructura para manejar el contexto del servidor.

Esta estructura contiene todos los elementos necesarios para el funcionamiento del servidor central, incluyendo el contexto CoAP, la configuración DTLS, la base de datos y el estado del servidor.

1.4.2 Field Documentation

1.4.2.1 ctx

```
coap_context_t* server_context_t::ctx
```

Contexto CoAP del servidor

1.4.2.2 db

```
sqlite3* server_context_t::db
```

Conexión a la base de datos SQLite

1.4.2.3 port

```
int server_context_t::port
```

Puerto de escucha del servidor

1.4.2.4 psk_file

```
char* server_context_t::psk_file
```

Ruta al archivo de claves PSK

1.4.2.5 running

```
int server_context_t::running
```

Flag de estado del servidor (1=activo, 0=detenido)

1.4.2.6 ssl_ctx

```
SSL_CTX* server_context_t::ssl_ctx
```

Contexto SSL para DTLS

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

- include/servidor_central/ **server_functions.h**

Chapter 2

File Documentation

2.1 include/servidor_central/dtls_common_config.h File Reference

Configuración común para DTLS-PSK en el servidor central.

Data Structures

- struct **dtls_config_t**
  Configuración de seguridad DTLS-PSK.

Functions

- int **init_ssl_context** (SSL_CTX **ssl_ctx, const **dtls_config_t** *config)
  Inicializa la configuración SSL para DTLS-PSK.
- void **cleanup_ssl_context** (SSL_CTX *ssl_ctx)
  Limpia y libera el contexto SSL.
- int **setup_dtls_session** (coap_context_t *ctx, SSL_CTX *ssl_ctx, int port)
  Configura una sesión CoAP con DTLS-PSK.
- int **psk_server_callback** (SSL *ssl, const char *identity, unsigned char *psk, unsigned int max_psk_len)
  Callback para autenticación PSK del servidor.
- int **validate_psk_key** (const char *identity, const char *psk, const char *psk_file)
  Valida una clave PSK contra el archivo de claves.
- int **get_dtls_config_from_env** (**dtls_config_t** *config)
  Obtiene la configuración DTLS desde variables de entorno.
- int **configure_dtls_session** (SSL *ssl, const **dtls_config_t** *config)
  Configura los parámetros de sesión DTLS.

2.1.1 Detailed Description

Configuración común para DTLS-PSK en el servidor central.

Author

Sistema de Control de Ascensores

Version

2.0

Date

2025

Este archivo contiene las configuraciones y constantes necesarias para la implementación de DTLS-PSK (Pre-↩ Shared Key) en el servidor central. Define los parámetros de seguridad, timeouts y configuraciones de sesión.

See also

server_functions.h (p. **??**)

psk_validator.h (p. **??**)

2.1.2 Function Documentation

2.1.2.1 cleanup_ssl_context()

```
void cleanup_ssl_context (
            SSL_CTX * ssl_ctx )
```

Limpia y libera el contexto SSL.

Parameters

in,out	*ssl_ctx*	Contexto SSL a liberar

Esta función libera todos los recursos asociados al contexto SSL:

- Libera el contexto SSL
- Limpia el caché de sesiones
- Libera memoria dinámica

Note

Debe ser llamada al finalizar para evitar memory leaks

See also

init_ssl_context (p. **??**)

2.1.2.2 configure_dtls_session()

```
int configure_dtls_session (
            SSL * ssl,
            const  dtls_config_t * config )
```

Configura los parámetros de sesión DTLS.

Parameters

in	*ssl*	Conexión SSL a configurar
in	*config*	Configuración DTLS a aplicar

Returns

0 en caso de éxito, -1 en caso de error

Esta función configura los parámetros de una sesión DTLS:

- Establece el timeout de sesión
- Configura el MTU para DTLS
- Establece el timeout de retransmisión
- Configura los parámetros de seguridad

Note

Debe ser llamada después de crear la sesión SSL

2.1.2.3 get_dtls_config_from_env()

```
int get_dtls_config_from_env (
             dtls_config_t * config )
```

Obtiene la configuración DTLS desde variables de entorno.

Parameters

out	*config*	Configuración DTLS a llenar

Returns

0 en caso de éxito, -1 en caso de error

Esta función lee la configuración DTLS desde variables de entorno:

- DTLS_PSK_FILE: Ruta al archivo de claves PSK

- DTLS_TIMEOUT: Timeout de sesión en segundos
- DTLS_MTU: MTU para DTLS en bytes
- DTLS_RETRANSMIT_TIMEOUT: Timeout de retransmisión
- DTLS_MAX_CONNECTIONS: Número máximo de conexiones
- DTLS_SESSION_CACHE_SIZE: Tamaño del caché de sesiones

Note

Si una variable no está definida, usa el valor por defecto

2.1.2.4 init_ssl_context()

```
int init_ssl_context (
            SSL_CTX ** ssl_ctx,
            const  dtls_config_t * config )
```

Inicializa la configuración SSL para DTLS-PSK.

Parameters

out	*ssl_ctx*	Contexto SSL a inicializar
in	*config*	Configuración DTLS a aplicar

Returns

0 en caso de éxito, -1 en caso de error

Esta función configura el contexto SSL para DTLS-PSK:

- Inicializa la biblioteca OpenSSL
- Crea el contexto SSL con método DTLS
- Configura los parámetros de seguridad PSK
- Establece los timeouts y límites de conexión
- Configura el caché de sesiones

Note

Debe ser llamada antes de crear sesiones SSL

See also

cleanup_ssl_context (p. **??**)

2.1.2.5 psk_server_callback()

```
int psk_server_callback (
            SSL * ssl,
            const char * identity,
            unsigned char * psk,
            unsigned int max_psk_len )
```

Callback para autenticación PSK del servidor.

Parameters

in	*ssl*	Conexión SSL
in	*identity*	Identidad del cliente
in	*psk*	Clave PSK del cliente
in	*max_psk_len*	Longitud máxima de clave PSK

Returns

1 si la autenticación es exitosa, 0 en caso contrario

Este callback es llamado durante el handshake DTLS:

- Recibe la identidad y clave PSK del cliente
- Valida la clave contra el archivo de claves
- Retorna el resultado de la validación
- Configura la clave PSK en la sesión SSL

Note

Esta función es llamada automáticamente por OpenSSL

See also

validate_psk (p. **??**)

2.1.2.6 setup_dtls_session()

```
int setup_dtls_session (
            coap_context_t * ctx,
            SSL_CTX * ssl_ctx,
            int port )
```

Configura una sesión CoAP con DTLS-PSK.

Parameters

in	*ctx*	Contexto CoAP del servidor
in	*ssl_ctx*	Contexto SSL configurado
in	*port*	Puerto de escucha

Returns

0 en caso de éxito, -1 en caso de error

Esta función configura la sesión CoAP para DTLS-PSK:

- Crea el endpoint de escucha DTLS
- Configura los parámetros de sesión
- Establece los callbacks de autenticación PSK
- Configura los timeouts de sesión

Note

Debe ser llamada después de inicializar el contexto SSL

2.1.2.7 validate_psk_key()

```
int validate_psk_key (
            const char * identity,
            const char * psk,
            const char * psk_file )
```

Valida una clave PSK contra el archivo de claves.

Parameters

in	*identity*	Identidad del cliente
in	*psk*	Clave PSK a validar
in	*psk_file*	Ruta al archivo de claves PSK

Returns

1 si la clave es válida, 0 en caso contrario

Esta función valida la autenticación PSK:

- Lee el archivo de claves PSK
- Busca la identidad del cliente
- Compara la clave proporcionada con la almacenada
- Retorna el resultado de la validación

Note

El archivo de claves debe contener pares identity:key

See also

psk_server_callback (p. **??**)

2.2 dtls_common_config.h

Go to the documentation of this file.

```
00001
00016 #ifndef DTLS_COMMON_CONFIG_H
00017 #define DTLS_COMMON_CONFIG_H
00018
00019 #include <openssl/ssl.h>
00020 #include <openssl/err.h>
00021 #include <coap3/coap.h>
00022
00023 #ifdef __cplusplus
00024 extern "C" {
00025 #endif
00026
00033 typedef struct {
00034     char *psk_file;
00035     int psk_timeout;
00036     int dtls_mtu;
00037     int retransmit_timeout;
00038     int max_connections;
00039     int session_cache_size;
00040 } dtls_config_t;
00041
00052 #define DEFAULT_DTLS_CONFIG { \
00053     .psk_file = "/app/psk_keys.txt", \
00054     .psk_timeout = 30, \
00055     .dtls_mtu = 1280, \
00056     .retransmit_timeout = 2, \
00057     .max_connections = 100, \
00058     .session_cache_size = 50 \
00059 }
00060
00066 #define DEFAULT_DTLS_PORT 5684
00067
00073 #define MAX_PSK_LENGTH 128
00074
00080 #define MAX_PSK_IDENTITY_LENGTH 64
00081
00087 #define NUM_PSK_KEYS 15000
00088
00107 int init_ssl_context(SSL_CTX **ssl_ctx, const dtls_config_t *config);
00108
00122 void cleanup_ssl_context(SSL_CTX *ssl_ctx);
00123
00141 int setup_dtls_session(coap_context_t *ctx, SSL_CTX *ssl_ctx, int port);
00142
00162 int psk_server_callback(SSL *ssl, const char *identity, unsigned char *psk,
00163                         unsigned int max_psk_len);
00164
00183 int validate_psk_key(const char *identity, const char *psk, const char *psk_file);
00184
00202 int get_dtls_config_from_env(dtls_config_t *config);
00203
00220 int configure_dtls_session(SSL *ssl, const dtls_config_t *config);
00221
00222 #ifdef __cplusplus
00223 }
00224 #endif
00225
00226 #endif /* DTLS_COMMON_CONFIG_H */
```

2.3 include/servidor_central/logging.h File Reference

Sistema de logging estructurado para el servidor central.

Data Structures

- struct **logging_config_t**

  Configuración del sistema de logging.

Enumerations

- enum **log_level_t** {
  LOG_LEVEL_DEBUG = 0 , **LOG_LEVEL_INFO** = 1 , **LOG_LEVEL_WARN** = 2 , **LOG_LEVEL_ERROR** = 3 ,
  LOG_LEVEL_CRIT = 4 }
  Niveles de logging disponibles.

Functions

- int **init_logging** (const **logging_config_t** ∗config, const char ∗log_file_path)
  Inicializa el sistema de logging.
- void **cleanup_logging** (void)
  Limpia y cierra el sistema de logging.
- void **set_log_level** (**log_level_t** level)
  Establece el nivel mínimo de logging.
- **log_level_t** **get_log_level** (void)
  Obtiene el nivel actual de logging.
- void **_log_message** (**log_level_t** level, const char ∗file, int line, const char ∗func, const char ∗format,...)
  Función interna para escribir logs.

2.3.1 Detailed Description

Sistema de logging estructurado para el servidor central.

Author

Sistema de Control de Ascensores

Version

2.0

Date

2025

Este archivo define el sistema de logging completo del servidor central, incluyendo macros para diferentes niveles de log, funciones de inicialización y configuración del sistema de logging.

See also

server_functions.h (p. **??**)
main.c (p. **??**)

2.3.2 Enumeration Type Documentation

2.3.2.1 log_level_t

`enum` **log_level_t**

Niveles de logging disponibles.

Define los diferentes niveles de logging del sistema, ordenados de menor a mayor prioridad.

Enumerator

LOG_LEVEL_DEBUG	Información detallada para debugging
LOG_LEVEL_INFO	Información general del sistema
LOG_LEVEL_WARN	Advertencias que no impiden funcionamiento
LOG_LEVEL_ERROR	Errores que afectan funcionalidad
LOG_LEVEL_CRIT	Errores críticos que pueden causar fallos

2.3.3 Function Documentation

2.3.3.1 _log_message()

```
void _log_message (
             log_level_t level,
            const char * file,
            int line,
            const char * func,
            const char * format,
             ...  )
```

Función interna para escribir logs.

Parameters

in	*level*	Nivel del log
in	*file*	Archivo donde se originó el log
in	*line*	Línea donde se originó el log
in	*func*	Función donde se originó el log
in	*format*	Formato del mensaje
in	...	Argumentos variables del mensaje

Esta función es llamada internamente por las macros de logging. No debe ser llamada directamente desde el código.

2.3.3.2 cleanup_logging()

```
void cleanup_logging (
            void  )
```

Limpia y cierra el sistema de logging.

Esta función limpia los recursos del sistema de logging:

- Cierra el archivo de log si está abierto
- Libera memoria dinámica
- Resetea la configuración

Note

Debe ser llamada al finalizar el programa

See also

init_logging (p. **??**)

2.3.3.3 get_log_level()

```
log_level_t get_log_level (
            void  )
```

Obtiene el nivel actual de logging.

Returns

Nivel actual de logging

Esta función retorna el nivel mínimo de logging actualmente configurado.

2.3.3.4 init_logging()

```
int init_logging (
            const  logging_config_t * config,
            const char * log_file_path )
```

Inicializa el sistema de logging.

Parameters

in	*config*	Configuración del sistema de logging
in	*log_file_path*	Ruta al archivo de log (NULL para stdout)

Returns

0 en caso de éxito, -1 en caso de error

Esta función inicializa el sistema de logging:

- Configura el nivel mínimo de logging
- Abre el archivo de log si se especifica
- Configura el formato de salida
- Inicializa las opciones de timestamp y colores

Note

Debe ser llamada antes de usar cualquier macro de logging

See also

cleanup_logging (p. **??**)

2.3.3.5 set_log_level()

```
void set_log_level (
            log_level_t level )
```

Establece el nivel mínimo de logging.

Parameters

in	*level*	Nuevo nivel mínimo de logging

Esta función permite cambiar dinámicamente el nivel mínimo de logging durante la ejecución del programa.

Note

Solo los logs con nivel >= al mínimo serán mostrados

2.4 logging.h

Go to the documentation of this file.

```
00001
00016 #ifndef LOGGING_H
00017 #define LOGGING_H
00018
00019 #include <stdio.h>
00020 #include <time.h>
00021 #include <stdarg.h>
00022
00023 #ifdef __cplusplus
00024 extern "C" {
00025 #endif
00026
00033 typedef enum {
00034     LOG_LEVEL_DEBUG = 0,
00035     LOG_LEVEL_INFO = 1,
00036     LOG_LEVEL_WARN = 2,
00037     LOG_LEVEL_ERROR = 3,
00038     LOG_LEVEL_CRIT = 4
00039 } log_level_t;
00040
00047 typedef struct {
00048     log_level_t min_level;
00049     FILE *log_file;
00050     int use_timestamps;
00051     int use_colors;
00052     int use_thread_id;
00053     char *log_format;
00054 } logging_config_t;
00055
00066 #define DEFAULT_LOGGING_CONFIG { \
00067     .min_level = LOG_LEVEL_INFO, \
00068     .log_file = NULL, \
00069     .use_timestamps = 1, \
00070     .use_colors = 1, \
00071     .use_thread_id = 1, \
```

```
00072     .log_format = NULL \
00073 }
00074
00092 int init_logging(const logging_config_t *config, const char *log_file_path);
00093
00105 void cleanup_logging(void);
00106
00117 void set_log_level(log_level_t level);
00118
00126 log_level_t get_log_level(void);
00127
00141 void _log_message(log_level_t level, const char *file, int line, const char *func,
00142                   const char *format, ...);
00143
00156 #define LOG_DEBUG(format, ...) \
00157     _log_message(LOG_LEVEL_DEBUG, __FILE__, __LINE__, __func__, format, ##__VA_ARGS__)
00158
00171 #define LOG_INFO(format, ...) \
00172     _log_message(LOG_LEVEL_INFO, __FILE__, __LINE__, __func__, format, ##__VA_ARGS__)
00173
00186 #define LOG_WARN(format, ...) \
00187     _log_message(LOG_LEVEL_WARN, __FILE__, __LINE__, __func__, format, ##__VA_ARGS__)
00188
00201 #define LOG_ERROR(format, ...) \
00202     _log_message(LOG_LEVEL_ERROR, __FILE__, __LINE__, __func__, format, ##__VA_ARGS__)
00203
00216 #define LOG_CRIT(format, ...) \
00217     _log_message(LOG_LEVEL_CRIT, __FILE__, __LINE__, __func__, format, ##__VA_ARGS__)
00218
00230 #define LOG_FUNC_ENTER(func_name, ...) \
00231     LOG_DEBUG("Entering %s", func_name)
00232
00244 #define LOG_FUNC_EXIT(func_name, return_value) \
00245     LOG_DEBUG("Exiting %s with return value %d", func_name, return_value)
00246
00257 #define LOG_VAR(var_name, var_value, format) \
00258     LOG_DEBUG("%s = " format, var_name, var_value)
00259
00260 #ifdef __cplusplus
00261 }
00262 #endif
00263
00264 #endif /* LOGGING_H */
```

2.5 include/servidor_central/psk_validator.h File Reference

Validador de claves PSK para autenticación DTLS.

Functions

- int **psk_validator_init** (const char *keys_file_path)

  Inicializa el validador de claves PSK.
- int **psk_validator_check_key** (const char *key, size_t key_len)

  Valida si una clave PSK está en la lista de claves válidas.
- int **psk_validator_get_key_for_identity** (const char *identity, uint8_t *key_buffer, size_t buffer_size)

  Obtiene una clave PSK válida para una identidad específica.
- int **psk_validator_get_key_by_index** (int index, uint8_t *key_buffer, size_t buffer_size)

  Obtiene una clave PSK por índice específico.
- void **psk_validator_cleanup** (void)

  Libera los recursos del validador de claves PSK.
- int **psk_validator_get_key_count** (void)

  Obtiene el número total de claves PSK disponibles.
- int **psk_validator_is_initialized** (void)

  Verifica si el validador está inicializado.

2.5.1 Detailed Description

Validador de claves PSK para autenticación DTLS.

Author

Sistema de Control de Ascensores

Version

2.0

Date

2025

Este archivo define las funciones para validar y gestionar claves PSK (Pre-Shared Keys) utilizadas en la autenticación DTLS-PSK del servidor central. El sistema utiliza un archivo de 15,000 claves únicas pre-generadas para garantizar la seguridad de las comunicaciones.

See also

dtls_common_config.h (p. **??**)

server_functions.h (p. **??**)

2.5.2 Function Documentation

2.5.2.1 psk_validator_check_key()

```
int psk_validator_check_key (
            const char * key,
            size_t key_len )
```

Valida si una clave PSK está en la lista de claves válidas.

Parameters

in	*key*	Clave PSK a validar
in	*key_len*	Longitud de la clave en bytes

Returns

1 si la clave es válida, 0 si no lo es

Esta función valida una clave PSK contra la lista de claves válidas:

- Busca la clave en la tabla hash de claves cargadas
- Compara la clave proporcionada con las almacenadas
- Retorna el resultado de la validación

Note

La búsqueda es O(1) gracias al uso de tabla hash

See also

psk_validator_get_key_for_identity (p. **??**)

Parameters

in	*key*	Clave PSK a validar
in	*key_len*	Longitud de la clave en bytes

Returns

1 si la clave es válida, 0 si no lo es

Esta función valida una clave PSK contra la lista de claves válidas:

- Verifica que haya claves cargadas en memoria
- Compara la clave proporcionada con cada clave almacenada
- Utiliza comparación exacta de longitud y contenido
- Retorna el resultado de la validación

Note

La búsqueda es lineal O(n) donde n es el número de claves
La función es thread-safe para lecturas concurrentes

See also

psk_validator_get_key_for_identity (p. **??**)

2.5.2.2 psk_validator_cleanup()

```
void psk_validator_cleanup (
            void  )
```

Libera los recursos del validador de claves PSK.

Esta función limpia todos los recursos asociados al validador:

- Cierra el archivo de claves si está abierto
- Libera la memoria de la tabla hash de claves
- Resetea el estado del validador
- Libera cualquier buffer interno

Note

Debe ser llamada al finalizar para evitar memory leaks

See also

psk_validator_init (p. **??**)

Esta función limpia todos los recursos asociados al validador:

- Libera cada clave individual del array
- Libera el array de punteros a claves
- Resetea los contadores y punteros
- Prepara el validador para una nueva inicialización

Note

Debe ser llamada al finalizar para evitar memory leaks
Es seguro llamar esta función múltiples veces

See also

psk_validator_init (p. **??**)

2.5.2.3 psk_validator_get_key_by_index()

```
int psk_validator_get_key_by_index (
            int index,
            uint8_t * key_buffer,
            size_t buffer_size )
```

Obtiene una clave PSK por índice específico.

Parameters

in	*index*	Índice de la clave en el archivo (0-based)
out	*key_buffer*	Buffer donde se almacenará la clave
in	*buffer_size*	Tamaño del buffer en bytes

Returns

0 si se obtuvo correctamente, -1 en caso de error

Esta función obtiene una clave PSK por su posición en el archivo:

- Lee la línea correspondiente al índice

- Parsea la clave del formato identity:key
- Copia la clave al buffer proporcionado
- Valida que el índice esté dentro del rango válido

Note

El índice debe estar entre 0 y NUM_PSK_KEYS-1

See also

psk_validator_get_key_for_identity (p. **??**)

Parameters

in	*index*	Índice de la clave en el archivo (0-based)
out	*key_buffer*	Buffer donde se almacenará la clave
in	*buffer_size*	Tamaño del buffer en bytes

Returns

0 si se obtuvo correctamente, -1 en caso de error

Esta función obtiene una clave PSK por su posición en el array:

- Valida que el índice esté dentro del rango válido
- Obtiene la clave del array de claves cargadas
- Copia la clave al buffer proporcionado
- Verifica que el buffer tenga espacio suficiente

Note

El índice debe estar entre 0 y count-1

See also

psk_validator_get_key_for_identity (p. **??**)

2.5.2.4 psk_validator_get_key_count()

```
int psk_validator_get_key_count (
            void  )
```

Obtiene el número total de claves PSK disponibles.

Returns

Número de claves PSK en el archivo

Esta función retorna el número total de claves PSK que han sido cargadas desde el archivo de claves.

Note

Solo es válida después de llamar a psk_validator_init

2.5.2.5 psk_validator_get_key_for_identity()

```
int psk_validator_get_key_for_identity (
            const char * identity,
            uint8_t * key_buffer,
            size_t buffer_size )
```

Obtiene una clave PSK válida para una identidad específica.

Parameters

in	*identity*	Identidad del cliente
out	*key_buffer*	Buffer donde se almacenará la clave
in	*buffer_size*	Tamaño del buffer en bytes

Returns

0 si se obtuvo correctamente, -1 en caso de error

Esta función obtiene la clave PSK correspondiente a una identidad:

- Busca la identidad en el archivo de claves
- Copia la clave al buffer proporcionado
- Verifica que el buffer tenga espacio suficiente
- Retorna el resultado de la operación

Note

El buffer debe tener al menos MAX_PSK_LENGTH bytes

See also

psk_validator_check_key (p. **??**)

Parameters

in	*identity*	Identidad del cliente
out	*key_buffer*	Buffer donde se almacenará la clave
in	*buffer_size*	Tamaño del buffer en bytes

Returns

0 si se obtuvo correctamente, -1 en caso de error

Esta función obtiene la clave PSK correspondiente a una identidad:

- Calcula un hash determinístico de la identidad

- Usa el hash como índice para seleccionar una clave
- Copia la clave seleccionada al buffer proporcionado
- Verifica que el buffer tenga espacio suficiente

Note

La selección es determinística: misma identidad = misma clave

El algoritmo usa hash simple para distribución uniforme

See also

psk_validator_check_key (p. **??**)

2.5.2.6 psk_validator_init()

```
int psk_validator_init (
            const char * keys_file_path )
```

Inicializa el validador de claves PSK.

Parameters

in	*keys_file_path*	Ruta al archivo de claves PSK

Returns

0 si se inicializó correctamente, -1 en caso de error

Esta función inicializa el sistema de validación de claves PSK:

- Abre y lee el archivo de claves PSK
- Carga las claves en memoria para acceso rápido
- Valida el formato del archivo de claves
- Configura el sistema de búsqueda de claves

Note

El archivo de claves debe contener pares identity:key

Debe ser llamada antes de usar cualquier función de validación

See also

psk_validator_cleanup (p. **??**)

Parameters

in	*keys_file_path*	Ruta al archivo de claves PSK

Returns

0 si se inicializó correctamente, -1 en caso de error

Esta función inicializa el sistema de validación de claves PSK:

- Abre el archivo de claves PSK especificado
- Cuenta el número de líneas en el archivo
- Asigna memoria dinámica para almacenar las claves
- Lee todas las claves del archivo y las almacena en memoria
- Cierra el archivo después de la carga

Note

El archivo de claves debe contener una clave por línea
La función maneja automáticamente la memoria dinámica

See also

psk_validator_cleanup (p. **??**)

2.5.2.7 psk_validator_is_initialized()

```
int psk_validator_is_initialized (
            void  )
```

Verifica si el validador está inicializado.

Returns

1 si está inicializado, 0 en caso contrario

Esta función verifica si el validador de claves PSK ha sido inicializado correctamente y está listo para su uso.

Note

Útil para verificar el estado antes de usar otras funciones

2.6 psk_validator.h

Go to the documentation of this file.

```
00001
00017 #ifndef PSK_VALIDATOR_H
00018 #define PSK_VALIDATOR_H
00019
00020 #include <stddef.h>
00021 #include <stdint.h>
00022
00023 #ifdef __cplusplus
00024 extern "C" {
00025 #endif
00026
00044 int psk_validator_init(const char* keys_file_path);
00045
00062 int psk_validator_check_key(const char* key, size_t key_len);
00063
00082 int psk_validator_get_key_for_identity(const char* identity, uint8_t* key_buffer, size_t buffer_size);
00083
00102 int psk_validator_get_key_by_index(int index, uint8_t* key_buffer, size_t buffer_size);
00103
00116 void psk_validator_cleanup(void);
00117
00128 int psk_validator_get_key_count(void);
00129
00140 int psk_validator_is_initialized(void);
00141
00142 #ifdef __cplusplus
00143 }
00144 #endif
00145
00146 #endif /* PSK_VALIDATOR_H */
```

2.7 include/servidor_central/server_functions.h File Reference

Funciones principales del servidor central de control de ascensores.

Data Structures

- struct **server_context_t**

  Estructura para manejar el contexto del servidor.

Functions

- int **init_server_context** (**server_context_t** ∗server_ctx, int port, const char ∗psk_file)

  Inicializa el contexto del servidor.
- void **cleanup_server_context** (**server_context_t** ∗server_ctx)

  Limpia y libera los recursos del contexto del servidor.
- int **run_server** (**server_context_t** ∗server_ctx)

  Inicia el bucle principal del servidor.
- int **setup_coap_resources** (coap_context_t ∗ctx)

  Configura los recursos CoAP del servidor.
- int **validate_psk** (const char ∗identity, const char ∗psk, const char ∗psk_file)

  Valida una clave PSK contra el archivo de claves.
- int **handle_floor_request** (coap_session_t ∗session, coap_pdu_t ∗request, coap_pdu_t ∗response, sqlite3 ∗db)

  Procesa una solicitud de asignación de ascensor.
- int **handle_cabin_request** (coap_session_t ∗session, coap_pdu_t ∗request, coap_pdu_t ∗response, sqlite3 ∗db)

  Procesa una solicitud específica de cabina.
- int **assign_elevator** (const char ∗edificio_id, int piso_origen, int piso_destino, sqlite3 ∗db, char ∗∗ascensor↩ _asignado, int ∗tiempo_estimado)

  Ejecuta el algoritmo de asignación óptima de ascensores.

2.7.1 Detailed Description

Funciones principales del servidor central de control de ascensores.

Author

Sistema de Control de Ascensores

Version

2.0

Date

2025

Este archivo contiene las declaraciones de las funciones principales del servidor central que maneja la comunicación CoAP/DTLS-PSK y la gestión de asignación de ascensores.

See also

main.c (p. **??**)

dtls_common_config.h (p. **??**)

logging.h (p. **??**)

2.7.2 Function Documentation

2.7.2.1 assign_elevator()

```
int assign_elevator (
            const char * edificio_id,
            int piso_origen,
            int piso_destino,
            sqlite3 * db,
            char ** ascensor_asignado,
            int * tiempo_estimado )
```

Ejecuta el algoritmo de asignación óptima de ascensores.

Parameters

in	*edificio_id*	ID del edificio
in	*piso_origen*	Piso de origen de la solicitud
in	*piso_destino*	Piso de destino de la solicitud
in	*db*	Conexión a la base de datos
out	*ascensor_asignado*	ID del ascensor asignado
out	*tiempo_estimado*	Tiempo estimado de llegada en segundos

Returns

0 en caso de éxito, -1 en caso de error

Este algoritmo implementa la lógica de asignación:

- Consulta todos los ascensores disponibles del edificio
- Calcula la distancia de cada ascensor al piso de origen
- Considera la dirección de movimiento y carga actual
- Selecciona el ascensor que minimice el tiempo de espera
- Retorna la asignación óptima

Note

El algoritmo prioriza ascensores libres sobre ocupados

2.7.2.2 cleanup_server_context()

```
void cleanup_server_context (
            server_context_t * server_ctx )
```

Limpia y libera los recursos del contexto del servidor.

Parameters

in,out	*server_ctx*	Puntero a la estructura de contexto del servidor

Esta función libera todos los recursos asociados al servidor:

- Cierra el contexto CoAP
- Libera la configuración SSL
- Cierra la conexión a la base de datos
- Libera memoria dinámica

Note

Debe ser llamada al finalizar el servidor para evitar memory leaks

See also

init_server_context (p. **??**)

2.7.2.3 handle_cabin_request()

```
int handle_cabin_request (
            coap_session_t * session,
            coap_pdu_t * request,
            coap_pdu_t * response,
            sqlite3 * db )
```

Procesa una solicitud específica de cabina.

Parameters

in	*session*	Sesión CoAP del cliente
in	*request*	Petición CoAP recibida
in	*response*	Respuesta CoAP a enviar
in	*db*	Conexión a la base de datos

Returns

0 en caso de éxito, -1 en caso de error

Esta función procesa las solicitudes de cabina específica:

- Valida que el ascensor solicitado esté disponible
- Verifica que el ascensor pueda atender la solicitud
- Asigna la tarea al ascensor específico
- Actualiza la base de datos
- Genera la respuesta de confirmación

Note

Esta función es llamada automáticamente por el handler CoAP

2.7.2.4 handle_floor_request()

```
int handle_floor_request (
            coap_session_t * session,
            coap_pdu_t * request,
            coap_pdu_t * response,
            sqlite3 * db )
```

Procesa una solicitud de asignación de ascensor.

Parameters

in	*session*	Sesión CoAP del cliente
in	*request*	Petición CoAP recibida
in	*response*	Respuesta CoAP a enviar
in	*db*	Conexión a la base de datos

Returns

0 en caso de éxito, -1 en caso de error

Esta función procesa las solicitudes de asignación:

- Valida el formato JSON de la petición
- Consulta el estado actual de los ascensores
- Ejecuta el algoritmo de asignación óptima
- Actualiza la base de datos con la nueva tarea
- Genera la respuesta JSON con la asignación

Note

Esta función es llamada automáticamente por el handler CoAP

2.7.2.5 init_server_context()

```
int init_server_context (
              server_context_t * server_ctx,
             int port,
             const char * psk_file )
```

Inicializa el contexto del servidor.

Parameters

out	*server_ctx*	Puntero a la estructura de contexto del servidor
in	*port*	Puerto en el que escuchará el servidor
in	*psk_file*	Ruta al archivo de claves PSK

Returns

0 en caso de éxito, -1 en caso de error

Esta función inicializa todos los componentes necesarios para el funcionamiento del servidor:

- Configura el contexto CoAP
- Inicializa la configuración DTLS-PSK
- Abre la conexión a la base de datos
- Configura los recursos CoAP

Note

La función debe ser llamada antes de iniciar el servidor

See also

cleanup_server_context (p. **??**)

2.7.2.6 run_server()

```
int run_server (
            server_context_t * server_ctx )
```

Inicia el bucle principal del servidor.

Parameters

in	*server_ctx*	Puntero a la estructura de contexto del servidor

Returns

0 en caso de éxito, -1 en caso de error

Esta función inicia el bucle principal del servidor que:

- Escucha conexiones entrantes
- Procesa solicitudes CoAP
- Maneja la autenticación DTLS-PSK
- Ejecuta los algoritmos de asignación de ascensores
- Responde a los clientes

Note

Esta función es bloqueante y solo retorna cuando el servidor se detiene

See also

init_server_context (p. **??**)

2.7.2.7 setup_coap_resources()

```
int setup_coap_resources (
            coap_context_t * ctx )
```

Configura los recursos CoAP del servidor.

Parameters

in	*ctx*	Contexto CoAP del servidor

Returns

0 en caso de éxito, -1 en caso de error

Esta función registra los endpoints CoAP disponibles:

- /peticion_piso: Para solicitudes de asignación de ascensor
- /peticion_cab: Para solicitudes específicas de cabina
- /.well-known/core: Para descubrimiento de recursos

Note

Debe ser llamada después de inicializar el contexto CoAP

2.7.2.8 validate_psk()

```
int validate_psk (
            const char * identity,
            const char * psk,
            const char * psk_file )
```

Valida una clave PSK contra el archivo de claves.

Parameters

in	*identity*	Identidad del cliente
in	*psk*	Clave PSK proporcionada por el cliente
in	*psk_file*	Ruta al archivo de claves PSK

Returns

1 si la clave es válida, 0 en caso contrario

Esta función valida la autenticación DTLS-PSK:

- Lee el archivo de claves PSK
- Busca la identidad del cliente
- Compara la clave proporcionada con la almacenada
- Retorna el resultado de la validación

Note

El archivo de claves debe contener pares identity:key

See also

psk_validator.h (p. **??**)

2.8 server_functions.h

Go to the documentation of this file.

```
00001
00016 #ifndef SERVER_FUNCTIONS_H
00017 #define SERVER_FUNCTIONS_H
00018
00019 #include <coap3/coap.h>
00020 #include <openssl/ssl.h>
00021 #include <openssl/err.h>
00022 #include <sqlite3.h>
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00035 typedef struct {
00036     coap_context_t *ctx;
00037     SSL_CTX *ssl_ctx;
00038     sqlite3 *db;
00039     int running;
00040     char *psk_file;
00041     int port;
00042 } server_context_t;
00043
00063 int init_server_context(server_context_t *server_ctx, int port, const char *psk_file);
00064
00079 void cleanup_server_context(server_context_t *server_ctx);
00080
00098 int run_server(server_context_t *server_ctx);
00099
00114 int setup_coap_resources(coap_context_t *ctx);
00115
00134 int validate_psk(const char *identity, const char *psk, const char *psk_file);
00135
00155 int handle_floor_request(coap_session_t *session, coap_pdu_t *request,
00156                          coap_pdu_t *response, sqlite3 *db);
00157
00177 int handle_cabin_request(coap_session_t *session, coap_pdu_t *request,
00178                          coap_pdu_t *response, sqlite3 *db);
00179
00201 int assign_elevator(const char *edificio_id, int piso_origen, int piso_destino,
00202                     sqlite3 *db, char **ascensor_asignado, int *tiempo_estimado);
00203
00204 #ifdef __cplusplus
00205 }
00206 #endif
00207
00208 #endif /* SERVER_FUNCTIONS_H */
```

2.9 src/main.c File Reference

Servidor Central del Sistema de Control de Ascensores con DTLS-PSK.

Functions

- static int **session_event_handler** (coap_session_t *session, const coap_event_t event)

  Callback para configurar sesiones DTLS con timeouts optimizados.
- static const coap_bin_const_t * **get_psk_info** (coap_bin_const_t *identity, coap_session_t *session, void *arg)

  Callback PSK personalizado para autenticación DTLS-PSK.
- void **handle_sigint** (int signum)

  Manejador de señal para SIGINT (Ctrl+C)
- void **generate_unique_task_id** (char *task_id_out, size_t len)

  Genera un ID único para una tarea de ascensor.
- static char * **select_optimal_elevator** (cJSON *elevadores_estado, int piso_origen, const char *direccion↩ _llamada)

Encuentra el ascensor más cercano para una llamada de piso.

- static void **hnd_floor_call** (coap_resource_t ∗resource, coap_session_t ∗session, const coap_pdu_↩ t ∗request, const coap_string_t ∗query, coap_pdu_t ∗response)

  Manejador CoAP para solicitudes de llamada de piso.
- static void **hnd_cabin_request** (coap_resource_t ∗resource, coap_session_t ∗session, const coap_pdu_t ∗request, const coap_string_t ∗query, coap_pdu_t ∗response)

  Manejador CoAP para solicitudes de cabina.
- int **main** (int argc, char ∗∗argv)

  Función principal del Servidor Central de Ascensores.

Variables

- static int **running** = 1

  Bandera global para controlar el bucle principal del servidor.

2.9.1 Detailed Description

Servidor Central del Sistema de Control de Ascensores con DTLS-PSK.

Author

Sistema de Control de Ascensores

Version

2.1

Date

2025

Este archivo implementa el servidor central que gestiona la asignación de tareas a ascensores en el sistema distribuido de control de ascensores. El servidor utiliza CoAP sobre DTLS-PSK para comunicaciones seguras con los API Gateways y proporciona endpoints RESTful para gestionar solicitudes de ascensores.

Funcionalidades principales:

- **Servidor CoAP DTLS**: Configuración de servidor CoAP con seguridad DTLS-PSK
- **Gestión de solicitudes**: Procesamiento de peticiones de piso y cabina
- **Algoritmos de asignación**: Lógica inteligente para asignar ascensores a tareas
- **Generación de IDs únicos**: Creación de identificadores únicos para tareas
- **Respuestas JSON**: Envío de respuestas estructuradas a los gateways
- **Logging detallado**: Sistema de registro para monitoreo y debugging
- **Gestión de sesiones**: Optimización de timeouts y reconexiones DTLS

Endpoints CoAP soportados:

- `POST /peticion_piso`: Solicitudes de llamada de piso desde botones externos
- `POST /peticion_cabina`: Solicitudes de cabina desde interior de ascensores

Algoritmo de asignación de ascensores: Utiliza el algoritmo de proximidad inteligente implementado en **select↩ _optimal_elevator()** (p. **??**):

- Filtra ascensores disponibles (disponible=true)
- Calcula distancia absoluta desde piso_origen a cada ascensor
- Selecciona todos los ascensores con distancia mínima
- En caso de empate, selecciona aleatoriamente para distribuir carga
- Garantiza asignación óptima basada en proximidad geográfica

Seguridad DTLS-PSK:

- Autenticación mutua usando claves precompartidas
- Cifrado de todas las comunicaciones
- Validación de identidades de clientes
- Gestión de sesiones con timeouts optimizados
- Prevención de ataques de repetición

Configuración de red:

- Puerto: 5684 (estándar CoAP-DTLS)
- Interfaz: 0.0.0.0 (todas las interfaces)
- Protocolo: UDP con DTLS

Gestión de memoria:

- Liberación automática de recursos al terminar
- Manejo seguro de strings y buffers
- Prevención de memory leaks

See also

servidor_central/logging.h (p. **??**)
servidor_central/dtls_common_config.h (p. **??**)
psk_validator.h (p. **??**)
coap3/coap.h
cJSON.h

2.9.2 Function Documentation

2.9.2.1 generate_unique_task_id()

```
void generate_unique_task_id (
            char * task_id_out,
            size_t len )
```

Genera un ID único para una tarea de ascensor.

Parameters

out	*task_id_out*	Buffer donde se almacenará el ID generado
in	*len*	Tamaño del buffer de salida en bytes

Esta función genera un identificador único para tareas de ascensor basado en el timestamp actual del sistema con precisión de milisegundos.

Formato del ID generado:
`"T_{segundos_unix}{milisegundos}"`

Ejemplo de ID:

- "T_1640995200123" donde:
  - T_: Prefijo identificador de tarea
  - 1640995200: Segundos desde epoch Unix
  - 123: Milisegundos (3 dígitos)

Características:

- Unicidad temporal garantizada
- Formato legible y ordenable
- Precisión de milisegundos
- Compatible con sistemas distribuidos

Limitaciones:

- No es thread-safe (para entornos multihilo usar sincronización)
- Dependiente del reloj del sistema
- Máximo 1,000 IDs por segundo (limitación de milisegundos)

Uso típico:
```
char task_id[32];
generate_unique_task_id(task_id, sizeof(task_id));
// task_id contiene "T_1640995200123"
```

Note

El buffer de salida debe tener al menos 32 caracteres

Para entornos multihilo considerar usar mutex o contadores atómicos

See also

gettimeofday()

snprintf()

2.9.2.2 get_psk_info()

```
static const coap_bin_const_t * get_psk_info (
            coap_bin_const_t * identity,
            coap_session_t * session,
            void * arg )  [static]
```

Callback PSK personalizado para autenticación DTLS-PSK.

Parameters

in	*identity*	Identidad del cliente DTLS
in	*session*	Sesión CoAP asociada
in	*arg*	Argumento de usuario (no usado)

Returns

Puntero a la clave PSK correspondiente o NULL si no se encuentra

Esta función implementa el callback de autenticación PSK para DTLS:

Proceso de autenticación:

1. Recibe la identidad del cliente desde el handshake DTLS
2. Valida que la identidad siga el patrón "Gateway_Client_∗"
3. Obtiene la clave PSK determinística basada en la identidad
4. Retorna la clave para completar el handshake DTLS

Patrones de identidad aceptados:

- "Gateway_Client_∗": Cualquier identidad que empiece con este prefijo
- Se rechazan identidades que no sigan el patrón

Algoritmo de clave determinística:

- Usa **psk_validator_get_key_for_identity()** (p. **??**) para obtener clave
- La misma identidad siempre produce la misma clave
- Garantiza consistencia entre servidor y cliente

Seguridad:

- Validación estricta de patrones de identidad
- Logging detallado de intentos de conexión
- Prevención de ataques de identidad falsa

Note

Esta función es llamada automáticamente por libcoap durante handshake DTLS

See also

psk_validator_get_key_for_identity() (p. **??**)
coap_bin_const_t

2.9.2.3 handle_sigint()

```
void handle_sigint (
            int signum )
```

Manejador de señal para SIGINT (Ctrl+C)

Parameters

in	*signum*	Número de señal recibida (se espera SIGINT = 2)

Esta función implementa el manejo elegante de la señal SIGINT:

Funcionalidad:

- Establece la bandera global 'running' a 0
- Permite que el bucle principal termine de forma controlada
- Registra el evento en el log del sistema
- Evita terminación abrupta del servidor

Flujo de terminación:

1. Usuario presiona Ctrl+C
2. Sistema envía SIGINT al proceso
3. Esta función establece running = 0
4. Bucle principal detecta el cambio y termina
5. Se ejecutan rutinas de limpieza en **main()** (p. **??**)

Seguridad:

- No realiza operaciones complejas en el handler
- Solo modifica la bandera de control
- Es thread-safe para el contexto de señales

Note

Esta función debe ser registrada con signal() o sigaction()
Solo modifica variables globales para evitar problemas de reentrancia

See also

running (p. **??**)
main() (p. **??**)

2.9.2.4 hnd_cabin_request()

```
static void hnd_cabin_request (
            coap_resource_t * resource,
            coap_session_t * session,
            const coap_pdu_t * request,
            const coap_string_t * query,
            coap_pdu_t * response )  [static]
```

Manejador CoAP para solicitudes de cabina.

Parameters

in	*resource*	Recurso CoAP que recibió la solicitud
in	*session*	Sesión CoAP del cliente que envió la solicitud
in	*request*	PDU de la solicitud CoAP recibida
in	*query*	Parámetros de consulta de la URI (no utilizado)
out	*response*	PDU de respuesta CoAP a enviar al cliente

Esta función procesa las solicitudes de cabina (cabin requests) recibidas desde los API Gateways. Las solicitudes de cabina provienen del interior de los ascensores cuando los usuarios presionan botones de destino.

Endpoint: `POST /peticion_cabina`

Formato JSON esperado:

```
{
  "id_edificio": "E1",
  "solicitando_ascensor_id": "E1A1",
  "piso_destino_solicitud": 8,
  "elevadores_estado": [
    {
      "id_ascensor": "E1A1",
      "piso_actual": 3,
      "estado_puerta": "CERRADA",
      "disponible": true,
      "tarea_actual_id": null,
      "destino_actual": null
    }
  ]
}
```

Respuesta JSON de éxito:

```
{
  "tarea_id": "T_1640995200456",
  "ascensor_asignado_id": "E1A1"
}
```

Códigos de respuesta HTTP:

- `200 OK`: Asignación exitosa
- `400 Bad Request`: JSON inválido o campos faltantes

Algoritmo de asignación: Para solicitudes de cabina, el ascensor asignado es siempre el mismo que realizó la solicitud (auto-asignación). Esto es lógico ya que la solicitud proviene del interior del ascensor.

Validaciones realizadas:

- Verificación de formato JSON válido
- Validación de campos obligatorios
- Comprobación de tipos de datos correctos
- Verificación de array de estado de ascensores válido

Diferencias con llamadas de piso:

- No requiere algoritmo de selección de ascensor
- El ascensor solicitante se auto-asigna
- No necesita verificar disponibilidad de otros ascensores

- Proceso más directo y eficiente

Gestión de errores:

- Logging detallado de errores y warnings
- Respuestas JSON con información de error
- Liberación automática de memoria en caso de error

Note

Esta función es llamada automáticamente por libcoap

No requiere algoritmo de selección de ascensor como las llamadas de piso

See also

generate_unique_task_id() (p. **??**)

RESOURCE_CABIN_REQUEST

cJSON_ParseWithLength()

2.9.2.5 hnd_floor_call()

```
static void hnd_floor_call (
            coap_resource_t * resource,
            coap_session_t * session,
            const coap_pdu_t * request,
            const coap_string_t * query,
            coap_pdu_t * response )  [static]
```

Manejador CoAP para solicitudes de llamada de piso.

Parameters

in	*resource*	Recurso CoAP que recibió la solicitud
in	*session*	Sesión CoAP del cliente que envió la solicitud
in	*request*	PDU de la solicitud CoAP recibida
in	*query*	Parámetros de consulta de la URI (no utilizado)
out	*response*	PDU de respuesta CoAP a enviar al cliente

Esta función procesa las solicitudes de llamada de piso (floor calls) recibidas desde los API Gateways. Implementa el algoritmo de asignación de ascensores para atender llamadas desde botones externos de los edificios.

Endpoint: `POST /peticion_piso`

Formato JSON esperado:

```
{
  "id_edificio": "E1",
  "piso_origen_llamada": 5,
  "direccion_llamada": "SUBIENDO",
  "elevadores_estado": [
    {
```

```
      "id_ascensor": "E1A1",
      "piso_actual": 3,
      "estado_puerta": "CERRADA",
      "disponible": true,
      "tarea_actual_id": null,
      "destino_actual": null
    }
  ]
}
```

Respuesta JSON de éxito:

```
{
  "tarea_id": "T_1640995200123",
  "ascensor_asignado_id": "E1A1"
}
```

Códigos de respuesta HTTP:

- `200 OK`: Asignación exitosa
- `400 Bad Request`: JSON inválido o campos faltantes
- `503 Service Unavailable`: No hay ascensores disponibles

Algoritmo de asignación: Utiliza el algoritmo de proximidad inteligente implementado en **select_optimal_↩elevator()** (p. **??**):

- Filtra ascensores disponibles (disponible=true)
- Calcula distancia absoluta desde piso_origen a cada ascensor
- Selecciona todos los ascensores con distancia mínima
- En caso de empate, selecciona aleatoriamente para distribuir carga
- Garantiza asignación óptima basada en proximidad

Validaciones realizadas:

- Verificación de formato JSON válido
- Validación de campos obligatorios
- Comprobación de tipos de datos correctos
- Verificación de disponibilidad de ascensores

Gestión de errores:

- Logging detallado de errores y warnings
- Respuestas JSON con información de error
- Liberación automática de memoria en caso de error

Note

Esta función es llamada automáticamente por libcoap

La memoria del ID del ascensor asignado debe ser liberada por el llamador

See also

select_optimal_elevator() (p. **??**)

generate_unique_task_id() (p. **??**)

RESOURCE_FLOOR_CALL

cJSON_ParseWithLength()

2.9.2.6 main()

```
int main (
            int argc,
            char ** argv )
```

Función principal del Servidor Central de Ascensores.

Parameters

in	*argc*	Número de argumentos de línea de comandos
in	*argv*	Array de argumentos de línea de comandos

Returns

EXIT_SUCCESS (0) si el servidor termina correctamente, EXIT_FAILURE (1) en caso de error

Esta función implementa el punto de entrada principal del servidor central de control de ascensores. Configura y ejecuta un servidor CoAP con DTLS-PSK que gestiona solicitudes de ascensores desde API Gateways.

Flujo de inicialización:

1. **Configuración de señales**: Registra manejador para SIGINT (Ctrl+C)
2. **Inicialización de libCoAP**: Configura logging y contexto CoAP
3. **Configuración de red**: Establece dirección y puerto de escucha
4. **Configuración DTLS-PSK**: Configura autenticación y cifrado
5. **Inicialización PSK**: Carga validador de claves precompartidas
6. **Registro de recursos**: Configura endpoints CoAP
7. **Bucle principal**: Procesa solicitudes hasta terminación

Configuración de seguridad:

- Autenticación DTLS-PSK con claves precompartidas
- Validación de identidades de clientes
- Cifrado de todas las comunicaciones
- Timeouts optimizados para estabilidad

Recursos CoAP registrados:

- `POST /peticion_piso`: Solicitudes de llamada de piso
- `POST /peticion_cabina`: Solicitudes de cabina

Gestión de errores:

- Validación de configuración de red

- Verificación de inicialización de componentes
- Logging detallado de errores críticos
- Terminación elegante en caso de fallo

Terminación elegante:

- Respuesta a señal SIGINT (Ctrl+C)
- Liberación de recursos de memoria
- Cierre de conexiones DTLS
- Limpieza de contexto CoAP

Configuración de red:

- Puerto: 5684 (estándar CoAP-DTLS)
- Interfaz: 0.0.0.0 (todas las interfaces)
- Protocolo: UDP con DTLS

Note

El servidor se ejecuta indefinidamente hasta recibir SIGINT

Requiere archivo de claves PSK para funcionamiento completo

See also

handle_sigint() (p. **??**)
session_event_handler() (p. **??**)
get_psk_info() (p. **??**)
psk_validator_init() (p. **??**)
hnd_floor_call() (p. **??**)
hnd_cabin_request() (p. **??**)

2.9.2.7 select_optimal_elevator()

```
static char * select_optimal_elevator (
            cJSON * elevadores_estado,
            int piso_origen,
            const char * direccion_llamada )  [static]
```

Encuentra el ascensor más cercano para una llamada de piso.

Parameters

in	*elevadores_estado*	Array JSON con el estado de todos los ascensores
in	*piso_origen*	Piso desde donde se realiza la llamada
in	*direccion_llamada*	Dirección solicitada ("up" o "down")

Returns

ID del ascensor asignado (debe liberarse con free()) o NULL si no hay ascensores disponibles

Esta función implementa un algoritmo de asignación inteligente que selecciona el ascensor más cercano al piso de origen de la llamada. Utiliza un algoritmo de proximidad optimizado para minimizar el tiempo de espera.

Algoritmo de selección:

1. **Filtrado inicial**: Solo considera ascensores disponibles (disponible == true)
2. **Cálculo de distancia**: Distancia absoluta entre piso_actual y piso_origen
3. **Búsqueda de mínimos**: Encuentra la distancia mínima entre todos los candidatos
4. **Resolución de empates**: Si hay múltiples ascensores con distancia mínima, selecciona aleatoriamente

Criterios de disponibilidad:

- Campo "disponible" debe ser true
- Campo "id_ascensor" debe ser string válido
- Campo "piso_actual" debe ser número válido

Ejemplo de funcionamiento:

```
Ascensores en pisos: [3, 6, 8, 10]
Llamada desde piso: 0
Distancias calculadas: [3, 6, 8, 10]
Resultado: Selecciona ascensor en piso 3 (distancia mínima = 3)
```

Gestión de memoria:

- Asigna memoria dinámicamente para el ID del ascensor seleccionado
- El llamador debe liberar la memoria con free()
- En caso de error, retorna NULL sin asignar memoria

Optimizaciones:

- Búsqueda en dos pasadas para eficiencia
- Uso de arrays temporales para candidatos
- Liberación automática de memoria no utilizada

Note

La función es thread-safe para lecturas concurrentes

El parámetro direccion_llamada no se usa actualmente pero se mantiene para futuras mejoras

See also

hnd_floor_call() (p. **??**)

cJSON_IsArray()

cJSON_GetArraySize()

2.9.2.8 session_event_handler()

```
static int session_event_handler (
            coap_session_t * session,
            const coap_event_t event )  [static]
```

Callback para configurar sesiones DTLS con timeouts optimizados.

Parameters

in	*session*	Sesión CoAP que se está configurando
in	*event*	Tipo de evento DTLS/CoAP

Returns

0 en todos los casos (éxito)

Esta función maneja eventos de sesión DTLS y configura parámetros optimizados para mejorar la estabilidad de las conexiones:

Eventos manejados:

- COAP_EVENT_SERVER_SESSION_NEW: Configura timeouts para nueva sesión
- COAP_EVENT_DTLS_CLOSED: Registra cierre de sesión DTLS
- COAP_EVENT_DTLS_ERROR: Registra errores DTLS
- COAP_EVENT_SERVER_SESSION_DEL: Registra eliminación de sesión

Configuración de timeouts:

- ACK timeout: 5 segundos
- Random factor: 1.5 (para evitar colisiones)
- Max retransmit: 4 intentos

Note

Esta función es llamada automáticamente por libcoap

See also

coap_session_set_ack_timeout()
coap_session_set_ack_random_factor()
coap_session_set_max_retransmit()

2.9.3 Variable Documentation

2.9.3.1 running

```
int running = 1  [static]
```

Bandera global para controlar el bucle principal del servidor.

Esta variable se establece a 0 por el manejador de señal SIGINT para indicar que el servidor debe terminar de manera elegante.

See also

handle_sigint() (p. **??**)

2.10 src/psk_validator.c File Reference

Implementación del validador de claves PSK para autenticación DTLS.

Data Structures

- struct **psk_valid_keys_t**
  Estructura para almacenar las claves PSK válidas.

Functions

- int **psk_validator_init** (const char ∗keys_file_path)
  Inicializa el validador de claves PSK.
- int **psk_validator_check_key** (const char ∗key, size_t key_len)
  Valida si una clave PSK está en la lista de claves válidas.
- int **psk_validator_get_key_for_identity** (const char ∗identity, uint8_t ∗key_buffer, size_t buffer_size)
  Obtiene una clave PSK válida para una identidad específica.
- int **psk_validator_get_key_by_index** (int index, uint8_t ∗key_buffer, size_t buffer_size)
  Obtiene una clave PSK por índice específico.
- void **psk_validator_cleanup** (void)
  Libera los recursos del validador de claves PSK.
- int **psk_validator_get_key_count** (void)
  Obtiene el número total de claves PSK disponibles.
- int **psk_validator_is_initialized** (void)
  Verifica si el validador está inicializado.

Variables

- static **psk_valid_keys_t g_valid_keys** = {NULL, 0, 0}
  Variable global que almacena las claves PSK válidas.

2.10.1 Detailed Description

Implementación del validador de claves PSK para autenticación DTLS.

Author

Sistema de Control de Ascensores

Version

2.0

Date

2025

Este archivo implementa el sistema de validación de claves PSK utilizado en la autenticación DTLS-PSK del servidor central. El sistema carga claves desde un archivo pre-generado y proporciona funciones para validar y obtener claves de forma determinística basada en la identidad del cliente.

See also

psk_validator.h (p. **??**)

dtls_common_config.h (p. **??**)

2.10.2 Function Documentation

2.10.2.1 psk_validator_check_key()

```
int psk_validator_check_key (
            const char * key,
            size_t key_len )
```

Valida si una clave PSK está en la lista de claves válidas.

Parameters

in	*key*	Clave PSK a validar
in	*key_len*	Longitud de la clave en bytes

Returns

1 si la clave es válida, 0 si no lo es

Esta función valida una clave PSK contra la lista de claves válidas:

- Verifica que haya claves cargadas en memoria
- Compara la clave proporcionada con cada clave almacenada
- Utiliza comparación exacta de longitud y contenido
- Retorna el resultado de la validación

Note

La búsqueda es lineal O(n) donde n es el número de claves
La función es thread-safe para lecturas concurrentes

See also

psk_validator_get_key_for_identity (p. **??**)

2.10.2.2 psk_validator_cleanup()

```
void psk_validator_cleanup (
            void  )
```

Libera los recursos del validador de claves PSK.

Esta función limpia todos los recursos asociados al validador:

- Libera cada clave individual del array
- Libera el array de punteros a claves
- Resetea los contadores y punteros
- Prepara el validador para una nueva inicialización

Note

Debe ser llamada al finalizar para evitar memory leaks

Es seguro llamar esta función múltiples veces

See also

psk_validator_init (p. **??**)

2.10.2.3 psk_validator_get_key_by_index()

```
int psk_validator_get_key_by_index (
            int index,
            uint8_t * key_buffer,
            size_t buffer_size )
```

Obtiene una clave PSK por índice específico.

Parameters

in	*index*	Índice de la clave en el archivo (0-based)
out	*key_buffer*	Buffer donde se almacenará la clave
in	*buffer_size*	Tamaño del buffer en bytes

Returns

0 si se obtuvo correctamente, -1 en caso de error

Esta función obtiene una clave PSK por su posición en el array:

- Valida que el índice esté dentro del rango válido
- Obtiene la clave del array de claves cargadas
- Copia la clave al buffer proporcionado
- Verifica que el buffer tenga espacio suficiente

Note

El índice debe estar entre 0 y count-1

See also

psk_validator_get_key_for_identity (p. **??**)

2.10.2.4 psk_validator_get_key_count()

```
int psk_validator_get_key_count (
            void  )
```

Obtiene el número total de claves PSK disponibles.

Returns

Número de claves PSK en el archivo

Esta función retorna el número total de claves PSK que han sido cargadas desde el archivo de claves.

Note

Solo es válida después de llamar a psk_validator_init

2.10.2.5 psk_validator_get_key_for_identity()

```
int psk_validator_get_key_for_identity (
            const char * identity,
            uint8_t * key_buffer,
            size_t buffer_size )
```

Obtiene una clave PSK válida para una identidad específica.

Parameters

in	*identity*	Identidad del cliente
out	*key_buffer*	Buffer donde se almacenará la clave
in	*buffer_size*	Tamaño del buffer en bytes

Returns

0 si se obtuvo correctamente, -1 en caso de error

Esta función obtiene la clave PSK correspondiente a una identidad:

- Calcula un hash determinístico de la identidad
- Usa el hash como índice para seleccionar una clave
- Copia la clave seleccionada al buffer proporcionado
- Verifica que el buffer tenga espacio suficiente

Note

La selección es determinística: misma identidad = misma clave

El algoritmo usa hash simple para distribución uniforme

See also

psk_validator_check_key (p. **??**)

2.10.2.6 psk_validator_init()

```
int psk_validator_init (
            const char * keys_file_path )
```

Inicializa el validador de claves PSK.

Parameters

in	*keys_file_path*	Ruta al archivo de claves PSK

Returns

0 si se inicializó correctamente, -1 en caso de error

Esta función inicializa el sistema de validación de claves PSK:

- Abre el archivo de claves PSK especificado
- Cuenta el número de líneas en el archivo
- Asigna memoria dinámica para almacenar las claves
- Lee todas las claves del archivo y las almacena en memoria
- Cierra el archivo después de la carga

Note

El archivo de claves debe contener una clave por línea
La función maneja automáticamente la memoria dinámica

See also

psk_validator_cleanup (p. **??**)

2.10.2.7 psk_validator_is_initialized()

```
int psk_validator_is_initialized (
            void  )
```

Verifica si el validador está inicializado.

Returns

1 si está inicializado, 0 en caso contrario

Esta función verifica si el validador de claves PSK ha sido inicializado correctamente y está listo para su uso.

Note

Útil para verificar el estado antes de usar otras funciones

2.10.3 Variable Documentation

2.10.3.1 g_valid_keys

```
psk_valid_keys_t g_valid_keys = {NULL, 0, 0}  [static]
```

Variable global que almacena las claves PSK válidas.

Esta variable global mantiene el estado del validador de claves PSK. Contiene todas las claves cargadas desde el archivo de configuración.