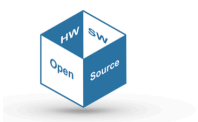
SE3recon: Securing IoT Communication Protocols

Project documentation

Release: April 3rd, 2019




HW
SW
Open
Source

Proprietary Notice

The following document offers information, which is subject to the terms and conditions described hereafter.

While care has been taken in preparing this document, some typographical errors, error or omissions may have occurred. We reserve the right to make changes to the content and information described herein or update such information at any time without notice. The opinions expressed are in good faith and while every care has been taken in preparing this document, some typographical errors, error or omissions may have occurred. We reserve the right to make changes to the content and information described herein or update such information at any time without notice. The opinion expressed are in good faith and while every care has been taken in preparing this document.

Authors

Giuseppe AIRÒ FARULLA *(CINI Cybersecurity National Lab)* giuseppe.airofarulla@polito.it

Mauro GUERRERA mauroguerrera92@gmail.com

Paolo PRINETTO *(President, CINI)* paolo.prinetto@polito.it

Annachiara RUOSPO ruospoannachiara@gmail.com

Trademarks

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by Blu5 View Pte Ltd. Other brands and names mentioned herein may be the trademarks of their respective owners. No use of these may be made for any purpose whatsoever without the prior written authorization of the owner company.

Disclaimer

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN "AS IS" BASIS AND ITS AUTHORS DISCLAIM ALL WARRANTIES, EXPRESS, OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY TAHT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PURPOSE.

THE SOFTWARE IS PROVIDED TO YOU "AS IS" AND WE MAKE NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER WITH RESPECT TO ITS FUNCTIONALITY, OPERABILITY, OR USE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PURPOSE, OR INFRINGEMENT. WE EXPRESSLY DISCLAIM ANY LIABILITY WHATSOEVER FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOSS REVENUES, LOST PROFITS, LOSSES RESULTING FROM BUSINESS INTERRUPTION OR LOSS OF DATA, REGARDLESS OF THE FORM OF ACTION OR LEGAL THEREUNDER WHICH THE LIABILITY MAY BE ASSERTED, EVEN IF ADVISED OF THE POSSIBILITY LIKELIHOOD OF SUCH DAMAGES.

Table of content

1 Introduction

SE3recon is a project aiming at securing communication between IoT devices, leveraging on the **SE***cube*™ platform.

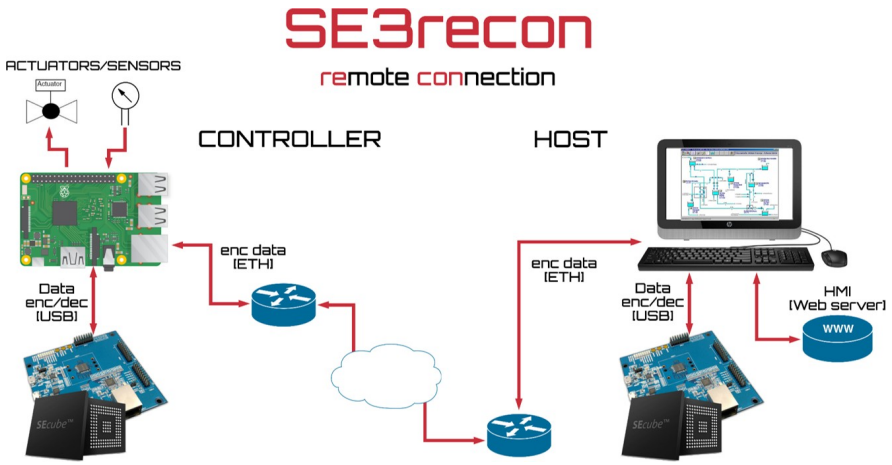


Figure 1.1: SE3recon platform

The graph shows the structure of SE3recon platform. The software architecture is based on a client-host structure: the host module sends control messages whereas the client module executes requested operations and replies with its current status. Communication between host and client is bidirectional, but typically it starts from host side, which sends a command to the controller and receives a message that reports client's status. All messages are encrypted and decrypted using **SE***cube*™ devices. Only encrypted data is sent through the network, ensuring a good level of security.

2 Design choices

All choices made in this project have been carefully evaluated. The first step was a deep analysis of all IoT communication protocols, possible target platforms and different ways of performing data encryption. Another part that has been evaluated was the implementation of a Human-Machine Interface used to control target platform.

2.1 Target platform - Raspberry Pi 3

Raspberry Pi 3 is the board chosen for simulating the target system. It can execute a full-fledged Linux OS, still having access to GPIOs. From the Hardware point of view, it is equipped with a 1.2GHz Quad-Core ARM Cortex-A53 Processor and a 1GB LPDDR2 memory but the device's key points are the low cost and low power. For this reason, it is mainly used in fields like IoT, Robotics, Media center, Server/cloud server, weather stations and Gaming. Furthermore, Raspberry Pi has high availability and high reliability; it is easy to configure and use and it is well supported with a good documentation. In our project it is the real leading actor of the system. To simulate target tasks, it has been decided to emulate physical devices using LEDs.

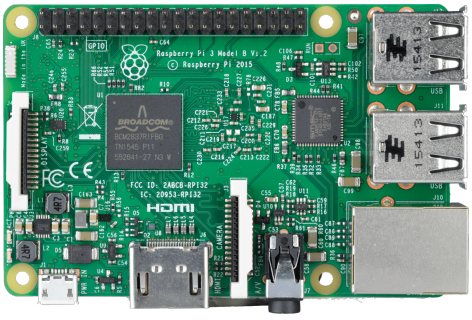
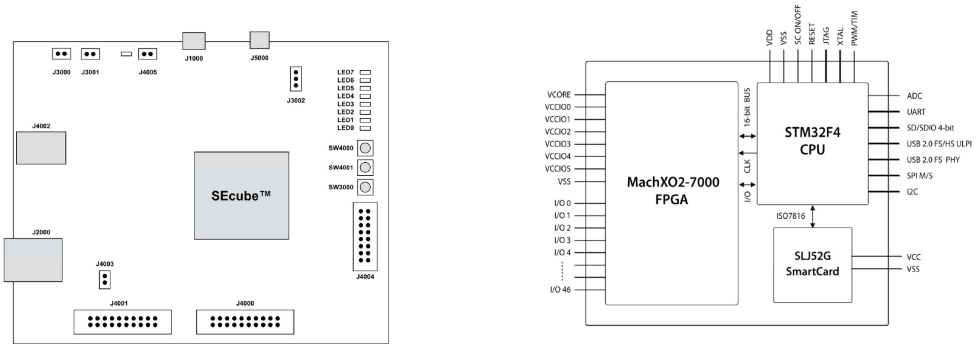


Figure 2.1: Raspberry Pi 3

2.2 SEcube™ DevKit



(a) SE*cube*™ DevKit (b) SE*cube*™ Chip

Figure 2.2: Hardware structure of the SE*cube*™ Devkit and of the SE*cube*™ Chip

The SE*cube*™ DevKit is an open development board designed to integrate the SE*cube*™ Chip in their hardware and software projects. The SE*cube*™ Chip is composed by three main components: a high-performance ARM Cortex M4 RISC CPU, produced by ST Microelectronics, STM32F4; an FPGA element, a Lattice MachX02 device, which is based on a fast-nonvolatile logic array; an EAL5+ certified security controller, hereafter named smartcard6, based on a secure chip produced by Infineon. The advantage of using this structure is security. SEFile provides functions for encrypting and decrypting. In this way, only encrypted messages are considered for the communications.

2.3 MQTT Protocol

MQTT is a machine-to-machine connectivity protocol mostly used for IoT. This is the protocol chosen for SE3recon to exchange data between the target system and host. Typically, it is used for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. It uses a Broker to manage communications between different channels called Topics. Each Broker manages different topics and every Topic can be dedicated to a specific group of sensors/actuators. MQTT protocol is designed as an extremely lightweight publish/subscribe messaging transport, actually it supports only three functions: Connect + Publish + Subscribe. For example, in a normal conversation, publisher sends (Publish) a message on a Topic (subject). All Subscribers listen (sub- scribe) for messages published on a specific Topic and activate the Broker, which is responsible of dispatching messages and looking for correct matching between Subscriber and Publisher (MQTT architecture shown in Figure 2.2).
MQTT can be extended with QoS but the main advantage is that it is light and fast. It is widely supported on Linux OS and there is a developer library called Mosquitto (http://mosquitto.org/).

2.4 Host Interface

Host side is managed by an Html page with a Javascript code which communicates thanks to a Socket with C++ code, running on the host side. From the Html page, user is able to press a Button for switching the On/Off LEDs on the Raspberry Pi.

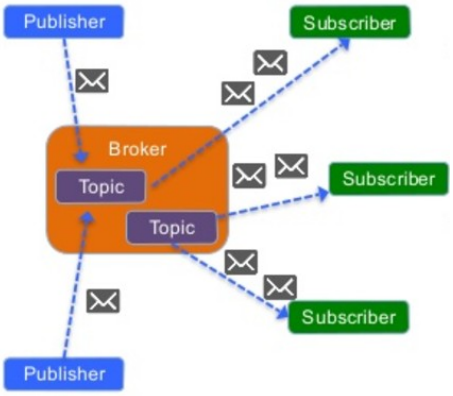


Figure 2.3: Mqtt protocol

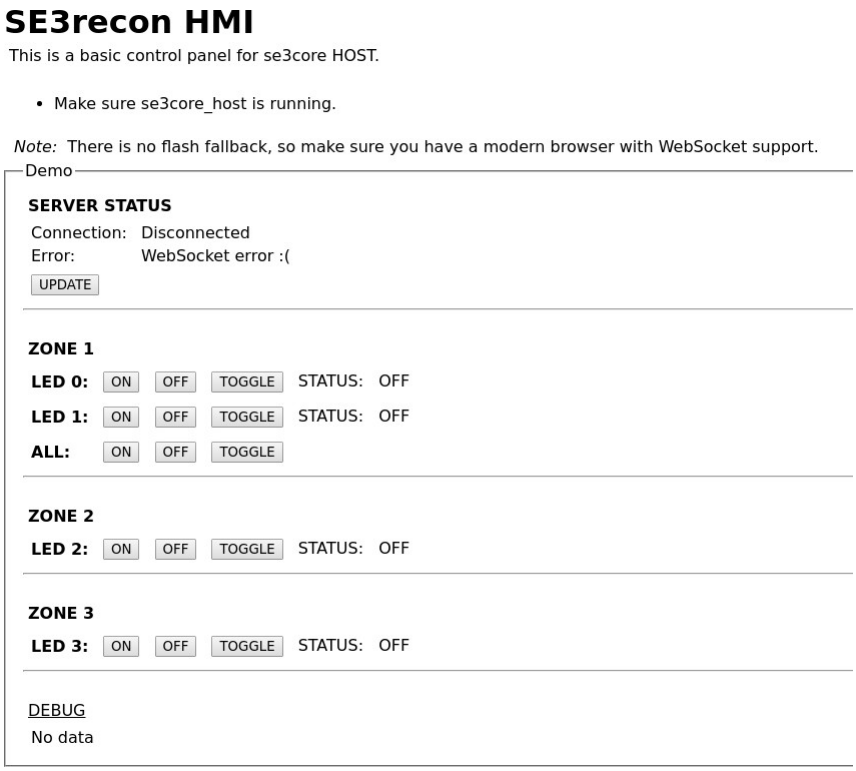
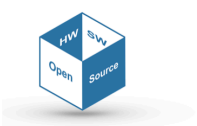


Figure 2.4: Html page


HW
SW
Open
Source

3 Development

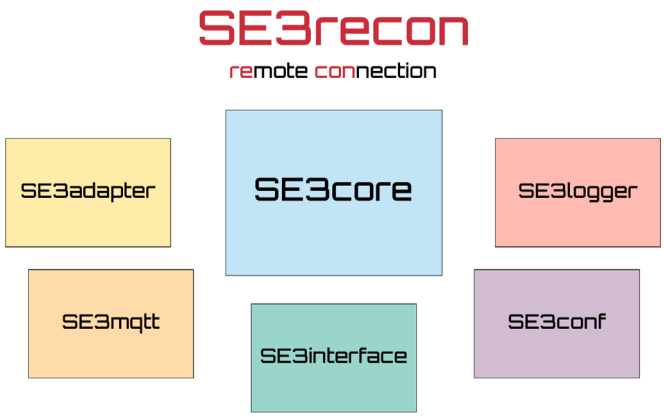


Figure 3.1: Software architecture

3.1 SE3core

SE3core is the main module of the architecture. It can be compiled into different versions, host and client applications, depending on the target system where it has to be executed. The core module includes different submodules, each of them is self-contained and has its own CMakeLists.txt file. Using cmake tool, each CMakeLists.txt is imported and submodule is automatically compiled.

3.1.1 Client

Client module includes:

- SE3adapter for controlling Raspberry's GPIO.
- MQTT Broker, useful for sending and receiving messages
- SE3interface: Module which allows to access the **SE***cube*™.

The client module is the application that runs on the target to monitor system's status and to perform requested operations. An MQTT control message is received from the host module, it is decrypted using the **SE***cube*™ board and eventually it is executed.
Once the task is completed, the client sends an encrypted message to the host, reporting its status. In order to perform all these operations, the client module includes the following submodules:

- se3adapter, provides class board to abstract hardware platform;
- se3interface, provides class se3interface to abstract **SE***cube*™;
- se3mqtt, provides class MQTT broker to wrap MQTT broker.

The host module provides an init function for each imported class. This allows to reduce code in the main function. For instance, the se3adapter submodule allows to create an object of type board, which provides an interface to hardware in/out pins. The related init function is board init, which instantiates a board object, sets up the hardware configuration and returns a boolean true value if all operations went fine.

Methods

- void mqtt_message_callback(struct mosquitto *mosq, void *obj, const struct mosquitto_message *message)

Description

Prototype of the MQTT message callback. This function will be executed every time the MQTT broker receives a message. se3mqtt has a test callback, but it is possible to replace it with a custom one.

Parameters

– struct mosquitto *mosq: mosquitto instance;
– void *obj: the user data provided in mosquitto new;
– onst struct mosquitto message *message: the message data. This variable and associated memory will be freed by the library after the callback completes. The client should make copies of any of the data it requires.

For further information see: http://mosquitto.org/api/files/mosquitto-h.html

- bool mqtt_init(void)

Description

This function calls all methods needed by the MQTT broker class to be initialized.

Return value

Returns true if MQTT broker is initialized correctly, false otherwise.

- bool secube_init(void)

Description

This function calls all methods needed by the se3interface class to be initialized.

Return value

Returns true if **SE***cube*™ is initialized correctly, false otherwise.

- bool board_init(void)

Description

This function calls all methods needed by the board class to be initialized.

Return value

Returns true if hardware peripheral is initialized correctly, false otherwise.

- void parse_message(uint8_t *l_dec_buffer, uint16_t l_dec_buffer_len)

Description

This function parses a message received from the host module through MQTT protocol and applies requested changes. It replies with a status message to host module.

Parameters

– uint8 t *l dec buffer: decoded buffer, received from MQTT and decrypted through **SE***cube*™. It will be parsed to determine which command has been issued by host;
– uint16 t l dec buffer len: length of decoded buffer.

- void update_status(void)

Description
This function calls the encryption function of SEcubeTM and sends the encrypted message to host through MQTT. The message reports current status of the platform (see Chapter 5 for further information about communication protocol).

3.1.2 Host

GHOst module includes:
- SE3interface: Module which allows to access the **SE***cube*™
- SE3mqtt
- SE3web: module useful for connecting with html page.

The host module is the application that sends control messages to the target system and displays the status of the target through a HMI web page. The main functions are creating a control message (in response to an event from the HMI), encrypting the message using the **SE***cube*™ board and sending the message to the target system through MQTT.
The client replies with its current status, which is decrypted and sent to the HMI.
In order to perform all these operations, the host module includes the following submodules:
- se3interface, provides class se3interface (5.1) to abstract the **SE***cube*™;
- se3mqtt, provides class mqtt broker (6.1) to wrap MQTT broker;
- se3web, provides class WebSocketServer, which is a wrapper for libwebsockets. The host module provides an init function for each imported class. This allows to reduce code in the main function. For instance, the se3mqtt submodule allows to create an object of type mqtt broker, which provides an interface to libmosquitto library. The related init function is mqtt init, which instantiates a MQTT broker object, sets up the object configuration and returns a boolean true value if all operations went fine.

Methods

- void mqtt_message_callback(struct mosquitto *mosq, void *obj, const struct mosquitto_message *message)

Description
Prototype of the MQTT message callback. This function will be executed every time the MQTT broker receives a message. se3mqtt has a test callback, but it is possible to replace it with a custom one.
Parameters
– struct mosquitto *mosq: mosquitto instance;
– void *obj: the user data provided in mosquitto new;
– const struct mosquitto message *message: the message data. This variable and asso- ciated memory will be freed by the library after the callback completes. The client should make copies of any of the data it requires.
For further information see: http://mosquitto.org/api/files/mosquitto-h.html

- bool mqtt_init(void)

Description
This function calls all methods needed by the MQTT broker class to be initialized.

Return value

Returns true if MQTT broker is initialized correctly, false otherwise.

- bool secube_init(void)

Description

This function calls all methods needed by the se3interface class to be initialized.

Return value

Returns true if **SE***cube*™ is initialized correctly, false otherwise.

- bool hmi_socket_init(void)

Description

This function calls all methods needed by the hmiSocket class to be initialized.

Return value

Returns true if hardware peripheral is initialized correctly, false otherwise.

- void update_hmi(uint8_t *l_dec_buffer, uint16_t l_dec_buffer_len)

Description

This function updates the HMI web page by sending a message on a socket. The message contains the system status, encoded using a custom protocol.

3.2 SE3adapter

It is used to configure the Hardware of the Client (Raspberry in our case). By default, there are four Pins configured, but it is possible to add other pins by calling the function digital pin add(). Then, with the functions digital write() and digital read(), pins can be written or read.

This class provides an abstraction level for the platform hardware. It allows to add handlers to digital and analog pins, wrapping low level calls to read/write functions.

Attributes

- int board_id;
- string board_vendor;
- string board_model;
- string board_serial;
- map<int16 t, digital pin*> digital pins: the map stores pointers to digital pin objects. Map key is the pin id;
- map<int16t,analogpin*> analogpins: the map stores pointers to analog pin objects. Map key is the pin id.

Methods

- board (int l_board_id, string l_board_vendor, string l_board_model, string l_board_serial)

Description
Constructor of the class board.
Parameters
– int l board id: ID of the board;
– string l board vendor: vendor of the board;
– string l board model: model of the board;
– string l board serial: serial number of the board.
Return value
It returns a pointer to an object of class board.

- ~board (void)

Description
Destructor of the class board.

- bool board_init(void)

Description
This function is used to initialize the wiringPi library.
Return value
Returns true if the initialization has been successful, false otherwise.

- int8_t digital_pin_add(int16_t l_pin_id, string l_pin_name, uint8_t l_pin_mode, uint8_t l_pin_value)

Description
This function is used to add a new digital pin handler to the the board.
Parameters
– int16 t l pin id: ID of the pin;
– string l pin name: name of the pin;
– uint8 t l pin mode: pin mode (input or output);
– uint8 t l pin value: initial pin value.
Return value
Returns STATUS OK if the pin has been added correctly, STATUS ERROR otherwise.

- int8_t digital_pin_remove(int16_t l_pin_id)

Description
This function is used to remove a digital pin handler from the board object.
Parameters
– int16 t l pin id: ID of the pin to be removed.
Return value
Returns STATUS OK if the pin has been removed correctly, STATUS ERROR otherwise.

- int8_t digital_pin_set_mode(int16_t l_pin_id, uint8_t l_pin_mode)

Description
This function is used to set digital pin mode (input or output).
Parameters
– int16 t l pin id: ID of the pin;
– uint8 t l pin mode: pin mode.

Return value
Returns STATUS OK if the pin mode has been set correctly, STATUS ERROR otherwise.

- int8_t analog_pin_add(int16_t l_pin_id, string l_pin_name, uint8_t l_pin_mode, uint16_t l_pin_value)

Description
This function is used to add a new analog pin handler to the the board.
Parameters
– int16 t l pin id: ID of the pin;
– string l pin name: name of the pin;
– uint8 t l pin mode: pin mode (input or output);
– uint16 t l pin value: initial pin value.
Return value
Returns STATUS OK if the pin has been added correctly, STATUS ERROR otherwise.

- int8_t analog_pin_remove(int16_t l_pin_id)

Description
This function is used to remove an analog pin handler from the board object.
Parameters
– int16 t l pin id: ID of the pin to be removed.
Return value
Returns STATUS OK if the pin has been removed correctly, STATUS ERROR otherwise.

- int8_t analog_pin_set_mode(int16_t l_pin_id, uint8_t l_pin_mode)

Description
This function is used to set analog pin mode (input or output).
Parameters
– int16 t l pin id: ID of the pin;
– uint8 t l pin mode: pin mode.
Return value
Returns STATUS OK if the pin mode has been set correctly, STATUS ERROR otherwise.

- int8_t digital_read(int16_t l_pin_id)

Description
This function is used to read the value of a digital pin on the board.
Parameters
– int16 t l pin id: ID of the pin.
Return value
Returns DVALUE LOW or DVALUE HIGH, according to current pin value. If read fails, the function returns STATUS ERROR.

- int8_t digital_write(int16_t l_pin_id, bool l_pin_value)

Description
This function is used to write a digital value on a pin.
Parameters
– int16 t l pin id: ID of the pin;
– bool l pin value: value to write on pin.
Return value
Returns STATUS OK if the pin has been written correctly, STATUS ERROR otherwise.

- int8_t digital_toggle(int16_t l_pin_id)

Description
This function is used to toggle the value of a digital pin.

Parameters
– int16 t l pin id: ID of the pin;
Return value
Returns STATUS OK if the pin has been written toggled, STATUS ERROR otherwise.

- uint16_t analog_read(int16_t l_pin_id)

Description
This function is used to read the value of an analog pin on the board.
Parameters
– int16 t l pin id: ID of the pin.
Return value
Returns the value read from the analog pin.

- uint16_t analog_write(int16_t l_pin_id, uint16_t l_pin_value)

Description
This function is used to write a value on an analog pin.
Parameters
– int16 t l pin id: ID of the pin;
– uint16 t l pin value: value to write on pin.
Return value
Returns the value written on the pin.

3.3 SE3mqtt

This module includes all the needed functions for instantiate the communication through MQTT protocol. Mosquitto library has been included in this module to correctly work with Linux. Here Broker is activated and all the Topic managed through Publish/Subscribe functions.

This class provides wrapper methods for libmosquitto. It is used to set up a MQTT broker that allows to send message across the network.

Attributes

- int broker id: broker ID;
- bool init success: set to true after initialization;
- bool is connected: set to true if connected;
- bool is run: set to true if broker is running;
- string broker name: broker name;
- string broker host: broker ip address;
- int broker port: MQTT communication port (default is 1883;
- int broker keepalive: broker timer that keeps connection alive;
- int broker qos: quality of service parameter;
- struct mosquitto *mosq: pointer to mosquitto structure (see http://mosquitto.org/api/files/ mosquitto-h.html for further information about libmosquitto data types).

Methods

- mqtt_broker(int l_broker_id, string l_broker_name, string l_broker_host, int l_broker_port)

Description
Constructor of the mqtt broker class.
Parameters
– int l broker id: ID of the broker.
– string l broker name: name of the broker.
– string l broker host: IP address of the broker.
– int l broker port: Port of the broker.

- ~mqtt_broker(void)

Description
Class destructor.

- bool broker_init(void)

Description
Method used to initialize the Broker. Must be called after constructor, before running broker.
Return value
Returns true if initialization has been performed correctly, false otherwise.

- bool broker_run(void)

Description
Method used to start the broker.
Return value
Returns true if broker has been started, false otherwise.

- bool broker_stop(void)

Description
Method used to stop the broker.
Return value
Returns true if broker has been stopped, false otherwise.

- int broker_connect(void)

Description
Method used to connect the broker.
Return value
Returns STATUS OK if broker has been connected, STATUS ERROR otherwise.

- int broker_disconnect(void)

Description
Method used to disconnect the broker.
Return value
Returns STATUS OK if broker has been disconnected, STATUS ERROR otherwise.

- int broker_subscribe(string l_topic)

Description
Method used to subscribe on a specific topic.
Parameters
– String l_topic: topic name.

Return value
Returns STATUS OK if broker subscribed correctly to specified topic, STATUS ERROR otherwise.

- int broker_unsubscribe(string l_topic)

Description
Method used to unsubscribe from a specific topic.
Parameters
– String l_topic: topic name.
Return value
Returns STATUS OK if broker unsubscribed correctly from specified topic, STATUS ERROR otherwise.

- int broker_publish(uint8_t* l_payload, int l_payload_len, string l_topic)

Description
Method used to publish a message on a specified topic.
Parameters
– uint8 t* l payload: pointer to uint8 t array that stores the message to be published;
– int l payload len: length of the message to be published;
– string l topic: name of the topic where the message should be published;
Return value
Returns STATUS OK if broker published the message correctly, STATUS ERROR otherwise.

- void broker_set_message_callback(void (*on_message) (struct mosquitto *, void *, const struct mosquitto_message *))

Description
This method is used to set the callback that is executed when a message is received.
Parameters
– on message: pointer to message callback function. For further information about message callback, see http://mosquitto.org/api/files/mosquitto-h.html.

3.4 SE3interface

SE3interface is used from both Host and Client in order to access **SE***cube*™ functions to encrypt and decrypt data.
This class provides wrapper methods for **SE***cube*™ board functions. It is used to encrypt and decrypt data to be sent over the network.

Attributes

- int secube id: ID of the board;
- bool init success: signals if initialization has been executed correctly;
- bool virtual se3: determines whether **SE***cube*™ hardware is physical or emulated;
- bool logged in: set to true if login has been already performed (either admin or user login);
- bool time set: set to true if time has been already set;
- uint8 t board serial[SE3 SERIAL SIZE]: board serial number;
- uint8 t pin admin[PIN LEN]: administrator pin;
- uint8 t pin user[PIN LEN]: user pin;

- uint8 t data key 1[DATA KEY LEN]: data key 1;
- uint8 t data key 2[DATA KEY LEN]: data key 2;
- uint8 t data key 3[DATA KEY LEN]: data key 3;
- uint8 t *encrypted data: pointer to encrypted data;
- uint8 t *decrypted data: pointer to decrypted data;
- se3_disco it it: **SE***cube*™ device iterator;
- se3_device dev: **SE***cube*™ device;
- se3_session s: current session;
- uint32 t session id: current session ID.

Methods

- se3interface(int l_secube_id)

Parameters
– int l secube id: ID of **SE***cube*™ device.
Description
Class constructor.

- ~se3interface(void)

Description
Destructor of se3interface class.

- bool interface_init()

Description
This method searches for a **SE***cube*™ device and once found, performs device initialization. Must be executed right after constructor.
Return value
Returns true if **SE***cube*™ has been initialized correctly, false otherwise.

- bool interface_init(bool l_virtual_se3)

Description
Overload of previous method. It searches for a **SE***cube*™ device and once found, performs device initialization. Must be executed right after constructor. It allows to specify explicitly whether to use a physical device or a virtual one.
Parameters
– bool l virtual se3: Parameter which allows to specify whether to use a virtual **SE***cube*™ or a physical one.
Return value
Returns true if **SE***cube*™ has been initialized correctly, false otherwise.

- bool set_time(void)

Description
This method sets time in **SE***cube*™ device. Must be called after login function.
Return value
Boolean value that informs whether the function has worked correctly.

- void print_serial_number(void)

Description

Method used to print board serial number.

- void print_pin_admin(void)

Description

Method used to print administrator pin.

- void print_pin_user(void)

Description

Method used to print user pin.

- void print_data_key_1(void)

Description

Method used to print Key1.

- void print_data_key_2(void)

Description

Method used to print Key2.

- void print_data_key_3(void)

Description

Method used to print Key3.

- void set_serial_number(uint8_t* v)

Description

Method used to set board serial number.

Parameters

– v: pointer to serial number (uint8 t array of size 32).

- void set_pin_admin(uint8_t *l_pin_admin)

Description

Method used to set administrator pin.

Parameters

– uint8_t *l pin admin: pointer to admin pin (uint8_t array of size 32).

- void set_pin_user(uint8_t *l_pin_user)

Description

Method used to set user pin.

Parameters

– uint8_t *l pin user: pointer to user pin (uint8_t array of size 32).

- void set_data_key_1(uint8_t *l_data_key)

Description

Method used to set data key 1.

Parameters

– uint8_t *l data key: pointer to Key1 (uint8_t array of size 32).

- void set_data_key_2(uint8_t *l_data_key)

Description

Method used to set data key 2.

Parameters

– uint8_t *l data key: pointer to Key2 (uint8_t array of size 32).

- void set_data_key_3(uint8_t *l_data_key)

Description

Method used to set data key 3.

Parameters

– uint8_t *l data key: pointer to Key3 (uint8_t array of size 32).

- bool login_admin(void)

Description

Method used to login as Administrator.

Return value

Returns true if login is successful, false otherwise.

- bool login_user(void)

Description

Function used to login as User.

Return value

Returns true if login is successful, false otherwise.

- bool logout(void)

Description

Method used to logout.

Return value

Returns true if logout is successful, false otherwise.

- uint8_t* encrypt_buffer(uint8_t* l_buffer, int l_buffer_size, uint16_t* l_enc_buffer_len)

Description

Method used to encrypt data.

Parameters

– uint8_t *l buffer: data to encrypt.

– int_l buffer size: size of buffer to encrypt.
– uint16_t *l_enc_buffer_len: pointer to integer value that stores encrypted data size. It is set by the method.

Return value

Returns a pointer to encrypted data if encryption has been performed correctly, a NULL pointer otherwise. l_enc_buffer_len is set to encrypted data length if encryption is successful, to 0 otherwise.

- uint8_t* decrypt_buffer(uint8_t* l_buffer, int l_buffer_size, uint16_t* l_dec_buffer_len)

Description

Method used to decrypt data.

Parameters

– uint8_t *l_buffer: data to decrypt.
– int_l buffer size: size of buffer to decrypt.
– uint16_t *l_dec_buffer_len: pointer to integer value that stores decrypted data size. It is set by the method.

Return value

Returns a pointer to decrypted data if decryption has been performed correctly, a NULL pointer otherwise. l_dec_buffer_len is set to decrypted data length if decryption is succesful, to 0 otherwise.

3.5 SE3web

It is the module which allows the communication between the host and the Web Page. It has been implemented by including a wrapper (see link below). This library has been a freely download and modified from https://github.com/mnisjk/cppWebSockets.

This class provides a wrapper for libwebsockets. It is used to open a socket communication and to exchange messages between the host and the HMI web page. The class has some virtual methods that must be overridden by a child class (hmiSocket in this project). WebSocketServer wrapper has been freely downloaded and modified from https://github.com/mnisjk/cppWebSockets.

Functions

- void run(uint64_t timeout = 50)

Description

This method calls lws service function of libwebsckets. It serves any pending websocket request. If no more requests are present, lws service returns after timeout elapses. This method calls lws service continuously, to prevent the function from returning. It blocks the calling thread.

Parameters

– uint64 t timeout = 50: timeout in milliseconds.

- void wait(uint64_t timeout = 50)

Description
Wrapper method for lws service function of libwebsckets. It serves any pending websocket request. If no more requests are present, lws service returns after timeout elapses. This method calls lws service once. Call to lws service is non-blocking.
Parameters
– uint64 t timeout = 50: timeout in milliseconds.

- void send(int socketID, string data)

Description
This method sends data on the socket specified.
Parameters
– int socketID: socket ID of the connection;
– string data: data to be sent on the socket connection.

- void broadcast(string data)

Description
This method sends data on all registered socket connections.
Parameters
– string data: data to be sent on the socket connection.

- virtual void onConnect(int socketID) = 0

Description
Virtual connection callback. Must be overridden by child class.

- virtual void onMessage(int socketID, const string& data) = 0

Description
Virtual message callback. Must be overridden by child class.

- virtual void onDisconnect(int socketID) = 0

Description
Virtual disconnection callback. Must be overridden by child class.

- virtual void onError(int socketID, const string& message) = 0

Description
Virtual error callback. Must be overridden by child class.

3.6 SE3config

This module aims at setting the configuration of the System.

3.7 SE3logger

Finally, the SE3logger is responsible for saving and printing Log messages of any kind.

4 Set up the Environment

4.1 System requirements

The whole system has been developed and tested on Linux platforms. The client module runs on a Raspberry Pi 3 equipped with Arch Linux ARM, available at: https://archlinuxarm.org/platforms/armv7/broadcom/raspberry-pi-2.

The host module has been tested on a standard Ubuntu 16.04 distribution.
It is advisable to use Ubuntu 16.04 to build the host application. For simplicity, the client application has been built directly on the Raspberry Pi board.

4.2 Dependencies

In order to setup the build environment on a Linux system, the following packages are required:

4.2.1 Host - Ubuntu 16.04

- build-essential (>=12.1)
- cmake (>=3.5.1)
- make (>=4.1-6)
- git (>=2.7.4)
- libmosquitto1 (>=1.4.8)
- libmosquitto-dev (>=1.4.8)
- mosquitto-clients (>=1.4.8)
- libwebsockets7 (>=1.7.1)
- libwebsockets-dev (>=1.7.1)
- openjdk-8-jdk (>=8u131)

4.2.2 Client - Arch Linux ARMv7

- gcc (>=7.1.1)
- gdb (>=8.0)
- cmake (>=3.8.2)
- make (>=4.2-1)
- git (>=2.13.2)
- mosquitto (>=1.4.12)

Important notes: mosquitto developer libraries are not available on Arch Linux ARM, but can be easily installed by downloading the ARM packages at
https://launchpad.net/ubuntu/xenial/armhf/libmosquitto1/1.4.8-1ubuntu0.16.04.1
https://launchpad.net/ubuntu/xenial/armhf/libmosquitto-dev/1.4.8-1ubuntu0.16.04.1

In addition, the wiringPi library is needed to access Raspberry specific hardware functions. The library can be downloaded, built and installed following the instructions at
http://wiringpi.com/download-and-install/

4.3 Download the SE3core source code

SE3core project is stored on Bitbucket. In order to download sources, move to a folder of your choice and run the following commands:

cd <folder>
git clone https://stark-dev@bitbucket.org/2017pr09/se3core.git cd se3core
git submodule update --init --recursive

This will clone and initialize the top-level repository and all submodules. The --recursive option is needed because the **SE***cube*™ libraries repository is included as a submodule into se3interface.

4.4 Configuration

After downloading source code, some configuration steps are needed in order to build the core application. First of all, the core module allows to build both the client and the host modules. It can also emulate **SE***cube*™ platform or physical hardware pins, if needed. To set desired options, edit CMakeLists.txt file with an editor of your choice.
The following options are available:

option(DEMO_MODE "Build in demo mode?" OFF)
Demo mode allows to emulate physical hardware pins. When this option is on, any call to read/write functions will only affect the pin objects, but will not result in any physical read/write.

option(EMULATE_SE3 "Emulate SE3 peripheral?" OFF)
This option is used to emulate the **SE***cube*™ hardware. The **SE***cube*™ object must be instantiated and initialized, but any call to encrypt/decrypt functions will return the same buffer.

option(BUILD_HOST "Build host target?" OFF)
This option enables the build of host application.

option(BUILD_CLIENT "Build client target?" OFF)
This option enables the build of client application.
Just set to ON the desired options before compiling the project.

There are other parameters available in the CMakeLists.txt file that must be configured before compiling the project:

set (MQTT_BROKER_ID_CLIENT "0")
The MQTT client ID can be configured. It is only for debug purposes and has no effect on the system's functionalities (can be left to 0).

set (MQTT_BROKER_NAME_CLIENT "client_broker")
It is possible to set a name to the MQTT broker. Again, it is only for logging and debug purposes.

set (MQTT_BROKER_IP_CLIENT "localhost")

This option is used to set the MQTT broker address. It is mandatory to set a valid IP in order to make the system work.

set (MQTT_BROKER_PORT_CLIENT "1883")
It is possible to decide which port will be used by the MQTT broker. Default port is 1883. The same options are available for the host module.
Important note: it is mandatory to set both client and host broker IPs to the same value.

4.5 Compilation

Once all code has been downloaded and settings are configured, it is possible to build the project. Each module has its own CMakeLists.txt file that will be automatically parsed by the cmake tool. cmake allows to check dependencies and to build a complex project linking required libraries and creating desired executables. It also allows to install binaries into specified folders.
In order to compile the project, run the following commands:

First of all, create a build directory to build all the project out of the source directory (this is useful to avoid messing up with the git repository):

```
mkdir build
```

Move to the build directory and run cmake tool

```
cd build cmake ..
```

cmake will check all dependencies, read desired options and it will set up the build environment. Once the process is completed, run compilation with make tool:

```
make
```

Once compilation is completed, the se3core binaries will be available into the build directory. Client name is se3core client and host name is se3core host. One or both executables will be found into the folder, according to options set in configuration step (see 2.4). Optionally, install binaries into system:

```
sudo make install
```

This command will install binaries into /usr/bin folder.

5 The communication protocol

Communication between client and host is based on a simple text-based protocol. The host needs to set the value of hardware pins on the target platform or to get the current status of the system.
The client module replies with the system status. Ingoing messages are decrypted using SEcube™ and parsed by the receiver module. Symmetrically, outgoing messages are created by each module and encrypted.

5.1 Host to Client

A basic set of commands is available to control the hardware platform from the host. Client accepts the following commands:

- set #<pin-id>#<pin-value>: this command is used to set a specific value on a digital pin. For instance, set #3#1 will set pin with ID 3 to HIGH value, while set #2#0 will set pin with ID 2 to LOW value.
- toggle #<pin-id>: this command is used to toggle the value on a digital pin. For instance,

toggle #4 will toggle the value on pin with ID 4 (either from LOW to HIGH or vice versa).

- status: this command is used to retrieve the current status of the target system. It has no parameters.

All commands con be issued from the HMI web page and are sent to the client module over MQTT.

5.2 Client to Host

The client module replies to each command received from the host with its current status. Client hardware pins are grouped by zone (this helps to send a single command to all pins within the same group). The format chosen to represent client status is the following:

```
#<zone>#<pin-id>#<pin-value>[#<pin-id>#<pin-value>]
```

where `zone` is the name of the group of the pins. It is followed by pairs of values where the first number is the ID of the pin and the second number represents pin value. For instance, assume that the group with ID 3 contains four pins: pin with ID 3 is `LOW`, pin with ID 5 is `HIGH`, pin with ID 6 is `HIGH` and pin with ID 8 is `LOW`. The string associated to group 3 status is

```
#zone3#3#0#5#1#6#1#8#0
```

The number of pairs (pin ID, pin value) depends on the number of pins on a specified group. Client module is able to parse any number of pairs, from one to generic n value.