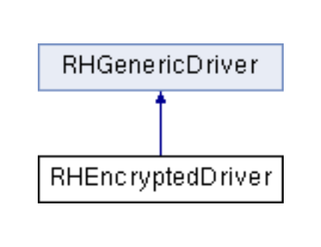
RHEncryptedDriver Class Reference

Virtual Driver to encrypt/decrypt data. Can be used with any other RadioHead driver. More...

Inheritance diagram for RHEncryptedDriver:



Public Member Functions

	RHEncryptedDriver (RHGenericDriver &driver, BlockCipher &blockcipher)
virtual bool	init ()
virtual bool	available ()
virtual bool	recv (uint8_t *buf, uint8_t *len)
virtual bool	send (const uint8_t *data, uint8_t len)
virtual uint8_t	maxMessageLength ()
virtual bool	waitPacketSent ()
virtual bool	waitPacketSent (uint16_t timeout)
virtual bool	waitAvailableTimeout (uint16_t timeout)
virtual bool	waitCAD ()
void	setCADTimeout (unsigned long cad_timeout)
virtual bool	isChannelActive ()
virtual void	setThisAddress (uint8_t thisAddress)
virtual void	setHeaderTo (uint8_t to)
virtual void	setHeaderFrom (uint8_t from)
virtual void	setHeaderId (uint8_t id)
virtual void	setHeaderFlags (uint8_t set, uint8_t clear=RH_FLAGS_APPLICATION_SPECIFIC)
virtual void	setPromiscuous (bool promiscuous)
virtual uint8_t	headerTo ()
virtual uint8_t	headerFrom ()
virtual uint8_t	headerId ()
virtual uint8_t	headerFlags ()
int16_t	lastRssi ()
RHMode	mode ()
void	setMode (RHMode mode) Sets the operating mode of the transport.
virtual bool	sleep ()
virtual uint16_t	rxBad ()
virtual uint16_t	rxGood ()
virtual uint16_t	txGood ()

▸ Public Member Functions inherited from RHGenericDriver

Additional Inherited Members

▸ Public Types inherited from RHGenericDriver
Defines different operating modes for the transport hardware. More...

▸ Static Public Member Functions inherited from RHGenericDriver

▸ Protected Attributes inherited from RHGenericDriver

Detailed Description

Virtual Driver to encrypt/decrypt data. Can be used with any other RadioHead driver.

This driver acts as a wrapper for any other RadioHead driver, adding encryption and decryption of messages that are passed to and from the actual radio driver. Only the message payload is encrypted, and not the to/from address or flags. Any of the encryption ciphers supported by ArduinoLibs Cryptographic Library http://rweather.github.io/arduinolibs/crypto.html may be used.

For successful communications, both sender and receiver must use the same cipher and the same key.

In order to enable this module you must uncomment #define RH_ENABLE_ENCRYPTION_MODULE at the bottom of RadioHead.h But ensure you have installed the Crypto directory from arduinolibs first: http://rweather.github.io/arduinolibs/index.html

Constructor & Destructor Documentation

◆ RHEncryptedDriver()

RHEncryptedDriver::RHEncryptedDriver (RHGenericDriver & driver,
BlockCipher & blockcipher
)

Constructor. Adds a ciphering layer to messages sent and received by the actual transport driver.

Parameters
- [in] **driver** The RadioHead driver to use to transport messages.
- [in] **blockcipher** The blockcipher (from arduinolibs) that crypt/decrypt data. Ensure that the blockcipher has had its key set before sending or receiving messages.

References RHGenericDriver::maxMessageLength().

Member Function Documentation

◆ available()

virtual bool RHEncryptedDriver::available ( ) inline virtual

Tests whether a new message is available from the Driver. On most drivers, this will also put the Driver into RHModeRx mode until a message is actually received by the transport, when it wil be returned to RHModeIdle. This can be called multiple times in a timeout loop

Returns
true if a new, complete, error-free uncollected message is available to be retreived by recv()

Implements RHGenericDriver.

References RHGenericDriver::available(), maxMessageLength(), recv(), and send().

◆ headerFlags()

virtual uint8_t RHEncryptedDriver::headerFlags ( ) inline virtual

Returns the FLAGS header of the last received message

Returns
The FLAGS header

Reimplemented from RHGenericDriver.

References RHGenericDriver::headerFlags().

◆ headerFrom()

virtual uint8_t RHEncryptedDriver::headerFrom ( ) inline virtual

Returns the FROM header of the last received message

Returns
The FROM header

Reimplemented from RHGenericDriver.

References RHGenericDriver::headerFrom().

◆ headerId()

virtual uint8_t RHEncryptedDriver::headerId ( ) inline virtual

Returns the ID header of the last received message

Returns
The ID header

Reimplemented from RHGenericDriver.

References RHGenericDriver::headerId().

◆ headerTo()

virtual uint8_t RHEncryptedDriver::headerTo ( ) inline virtual

Returns the TO header of the last received message

Returns
The TO header

Reimplemented from RHGenericDriver.

References RHGenericDriver::headerTo().

◆ init()

virtual bool RHEncryptedDriver::init ( ) inline virtual

Calls the real driver's init()

Returns
The value returned from the driver init() method;

Reimplemented from RHGenericDriver.

References RHGenericDriver::init().

◆ isChannelActive()

virtual bool RHEncryptedDriver::isChannelActive ( ) inline virtual

Determine if the currently selected radio channel is active. This is expected to be subclassed by specific radios to implement their Channel Activity Detection if supported. If the radio does not support CAD, returns true immediately. If a RadioHead radio supports isChannelActive() it will be documented in the radio specific documentation. This is called automatically by waitCAD().

Returns
true if the radio-specific CAD (as returned by override of isChannelActive()) shows the current radio channel as active, else false. If there is no radio-specific CAD, returns false.

Reimplemented from RHGenericDriver.

References RHGenericDriver::isChannelActive().

◆ lastRssi()

int16_t RHEncryptedDriver::lastRssi ( ) inline virtual

Returns the most recent RSSI (Receiver Signal Strength Indicator). Usually it is the RSSI of the last received message, which is measured when the preamble is received. If you called readRssi() more recently, it will return that more recent value.

Returns
The most recent RSSI measurement in dBm.

Reimplemented from RHGenericDriver.

References RHGenericDriver::lastRssi().

◆ maxMessageLength()

uint8_t RHEncryptedDriver::maxMessageLength ( ) virtual

Returns the maximum message length available in this Driver, which depends on the maximum length supported by the underlying transport driver.

Returns
The maximum legal message length

Implements RHGenericDriver.

References RHGenericDriver::maxMessageLength().

Referenced by available(), and send().

◆ mode()

RHMode RHEncryptedDriver::mode ( ) inline virtual

Returns the operating mode of the library.

Returns
the current mode, one of RF69_MODE_*

Reimplemented from RHGenericDriver.

References RHGenericDriver::mode().

◆ recv()

bool RHEncryptedDriver::recv (uint8_t * buf,
uint8_t * len
) virtual

Turns the receiver on if it not already on. If there is a valid message available, copy it to buf and return true else return false. If a message is copied, *len is set to the length (Caution, 0 length messages are permitted). You should be sure to call this function frequently enough to not miss any messages It is recommended that you call it in your main loop.

Parameters
- [in] **buf** Location to copy the received message
- [in,out] **len** Pointer to available space in buf. Set to the actual number of octets copied.

Returns
true if a valid message was copied to buf

Implements RHGenericDriver.

References RHGenericDriver::recv().

Referenced by available().

◆ rxBad()

virtual uint16_t RHEncryptedDriver::rxBad ( ) inline virtual

Returns the count of the number of bad received packets (ie packets with bad lengths, checksum etc) which were rejected and not delivered to the application. Caution: not all drivers can correctly report this count. Some underlying hardware only report good packets.

Returns
The number of bad packets received.

Reimplemented from RHGenericDriver.

References RHGenericDriver::rxBad().

◆ rxGood()

virtual uint16_t RHEncryptedDriver::rxGood ( ) inline virtual

Returns the count of the number of good received packets

Returns
The number of good packets received.

Reimplemented from RHGenericDriver.

References RHGenericDriver::rxGood().

◆ send()

bool RHEncryptedDriver::send (const uint8_t * data,
uint8_t len
) virtual

Waits until any previous transmit packet is finished being transmitted with waitPacketSent(). Then optionally waits for Channel Activity Detection (CAD) to show the channnel is clear (if the radio supports CAD) by calling waitCAD(). Then loads a message into the transmitter and starts the transmitter. Note that a message length of 0 is permitted.

Parameters
- [in] **data** Array of data to be sent
- [in] **len** Number of bytes of data to send specify the maximum time in ms to wait. If 0 (the default) do not wait for CAD before transmitting.

Returns
true if the message length was valid and it was correctly queued for transmit. Return false if CAD was requested and the CAD timeout timed out before clear channel was detected.

Implements RHGenericDriver.

References maxMessageLength(), and RHGenericDriver::send().

Referenced by available().

◆ setCADTimeout()

void RHEncryptedDriver::setCADTimeout (unsigned long cad_timeout) inline

Sets the Channel Activity Detection timeout in milliseconds to be used by waitCAD(). The default is 0, which means do not wait for CAD detection. CAD detection depends on support for isChannelActive() by your particular radio.

References RHGenericDriver::setCADTimeout().

◆ setHeaderFlags()

virtual void RHEncryptedDriver::setHeaderFlags (uint8_t set,
uint8_t clear = RH_FLAGS_APPLICATION_SPECIFIC
) inline virtual

Sets and clears bits in the FLAGS header to be sent in all subsequent messages First it clears he FLAGS according to the clear argument, then sets the flags according to the set argument. The default for clear always clears the application specific flags.

Parameters
- [in] **set** bitmask of bits to be set. Flags are cleared with the clear mask before being set.
- [in] **clear** bitmask of flags to clear. Defaults to RH_FLAGS_APPLICATION_SPECIFIC which clears the application specific flags, resulting in new application specific flags identical to the set.

Reimplemented from RHGenericDriver.

References RHGenericDriver::setHeaderFlags().

◆ setHeaderFrom()

virtual void RHEncryptedDriver::setHeaderFrom (uint8_t from) inline virtual

Sets the FROM header to be sent in all subsequent messages

Parameters
- [in] **from** The new FROM header value

Reimplemented from RHGenericDriver.

References RHGenericDriver::setHeaderFrom().

◆ setHeaderId()

virtual void RHEncryptedDriver::setHeaderId (uint8_t id) inline virtual

Sets the ID header to be sent in all subsequent messages

Parameters
- [in] **id** The new ID header value

Reimplemented from RHGenericDriver.

References RHGenericDriver::setHeaderId().

◆ setHeaderTo()

virtual void RHEncryptedDriver::setHeaderTo (uint8_t to) inline virtual

Sets the TO header to be sent in all subsequent messages

Parameters
- [in] **to** The new TO header value

Reimplemented from RHGenericDriver.

References RHGenericDriver::setHeaderTo().

◆ setPromiscuous()

virtual void RHEncryptedDriver::setPromiscuous (bool promiscuous) inline virtual

Tells the receiver to accept messages with any TO address, not just messages addressed to thisAddress or the broadcast address

Parameters
- [in] **promiscuous** true if you wish to receive messages with any TO address

Reimplemented from RHGenericDriver.

References RHGenericDriver::setPromiscuous().

◆ setThisAddress()

virtual void RHEncryptedDriver::setThisAddress (uint8_t thisAddress) inline virtual

Sets the address of this node. Defaults to 0xFF. Subclasses or the user may want to change this. This will be used to test the adddress in incoming messages. In non-promiscuous mode, only messages with a TO header the same as thisAddress or the broadcast addess (0xFF) will be accepted. In promiscuous mode, all messages will be accepted regardless of the TO header. In a conventional multinode system, all nodes will have a unique address (which you could store in EEPROM). You would normally set the header FROM address to be the same as thisAddress (though you dont have to, allowing the possibilty of address spoofing).

Parameters
- [in] **thisAddress** The address of this node.

Reimplemented from RHGenericDriver.

References RHGenericDriver::setThisAddress().

◆ sleep()

virtual bool RHEncryptedDriver::sleep ( ) inline virtual

Sets the transport hardware into low-power sleep mode (if supported). May be overridden by specific drivers to initialte sleep mode. If successful, the transport will stay in sleep mode until woken by changing mode it idle, transmit or receive (eg by calling send(), recv(), available() etc)

Returns
true if sleep mode is supported by transport hardware and the RadioHead driver, and if sleep mode was successfully entered. If sleep mode is not suported, return false.

Reimplemented from RHGenericDriver.

References RHGenericDriver::sleep().

◆ txGood()

virtual uint16_t RHEncryptedDriver::txGood ( ) inline virtual

Returns the count of the number of packets successfully transmitted (though not necessarily received by the destination)

Returns
The number of packets successfully transmitted

Reimplemented from RHGenericDriver.

References RHGenericDriver::txGood().

◆ waitAvailableTimeout()

virtual bool RHEncryptedDriver::waitAvailableTimeout (uint16_t timeout) inline virtual

Starts the receiver and blocks until a received message is available or a timeout

Parameters
- [in] **timeout** Maximum time to wait in milliseconds.

Returns
true if a message is available

Reimplemented from RHGenericDriver.

References RHGenericDriver::waitAvailableTimeout().

◆ waitCAD()

virtual bool RHEncryptedDriver::waitCAD ( ) inline virtual

Calls the waitCAD method in the driver

Returns
The return value from teh drivers waitCAD() method

Reimplemented from RHGenericDriver.

References RHGenericDriver::waitCAD().

◆ waitPacketSent() [1/2]

virtual bool RHEncryptedDriver::waitPacketSent ( ) inline virtual

Blocks until the transmitter is no longer transmitting.

Reimplemented from RHGenericDriver.

References RHGenericDriver::waitPacketSent().

◆ waitPacketSent() [2/2]

virtual bool RHEncryptedDriver::waitPacketSent (uint16_t timeout) inline virtual

Blocks until the transmitter is no longer transmitting. or until the timeout occuers, whichever happens first

Parameters
- [in] **timeout** Maximum time to wait in milliseconds.

Returns
true if the radio completed transmission within the timeout period. False if it timed out.

Reimplemented from RHGenericDriver.

References RHGenericDriver::waitPacketSent().

The documentation for this class was generated from the following files:

- RHEncryptedDriver.h
- RHEncryptedDriver.cpp

Generated by doxygen 1.8.13