

# RFM69 Session Key Library

---

Library Reference: [https://github.com/dewoodruff/RFM69\\_SessionKey](https://github.com/dewoodruff/RFM69_SessionKey)  
Document Version: 0.5  
Save Date: 4 October 2017

## Introduction

Enhancement of the standard RFM69 library (<https://github.com/LowPowerLab/RFM69>) by adding functions to exchange secured data messages between RFM69 transceivers. It is using one times session key handshake scheme developed by Dan Woodruff.

This library is to be used concurrently with a special version of the RFM69 Library:

(<https://github.com/TomWS1/RFM69/tree/virtualized>). The actual one used is updated version covering the latest changes of the standard library of March 2015.

---

### Notes:

1. The RFM69 library of 2<sup>nd</sup> of October 2015, includes the function virtualization, however, the extension of the control Bytes used by the Session Key library need to be added (see Appendix: New Control Bytes on page 13).
2. The RFM69 library was modified to be compatible with the ESP-32 processors and known as RFM69X library, the corresponding virtualized RFM69\_SeesionKey is known as RFM69X\_Session Library. This modification introduces the usage of the interruptHandling function, which is document in the flow diagram in appendix

---

The RFM69 SessionKey library described hereunder is a modified version of the original one, including the following improvements:

1. One Byte Random Session Key is replaced by a 4 Bytes Session Key as a snapshot of the system time at the moment of the session request
2. Some new public variables are defined for standardization
  - SESSION\_KEY\_LENGTH 4
  - RF69\_HEADER\_LENGTH 4
  - SESSION\_HEADER\_LENGTH RF69\_HEADER\_LENGTH + SESSION\_KEY\_LENGTH
  - SESSION\_MAX\_DATA\_LEN RF69\_MAX\_DATA\_LEN - SESSION\_KEY\_LENGTH
3. A test is done to avoid starting a session with the Broadcast node
4. A test is done at the session receiver to avoid receiving session data when the node is in promiscuous mode
5. New function (useSession3Acks) allowing 3 ACKs to be sent by the receiver at the end of the session instead of one
6. New function (session3AcksEnabled) allowing to check the Session3Acks value
7. New function (sessionWaitTime) allowing to change the SESSION KEY request response time watchdog
8. New function (sessionRespDelay) allowing to change SESSION KEY response delay for slow remote node
9. New public variable SESSION\_KEY\_RCV\_STATUS to be able to evaluate a Session negotiation failure, the following values are currently foreseen:
  - SESSION\_KEY\_RCV\_STATUS = 1 Receiver: A session key is requested and send
  - SESSION\_KEY\_RCV\_STATUS = 2 Sender: A session key is received an computed
  - SESSION\_KEY\_RCV\_STATUS = 3 Receiver: The received session key doesn't match with the expected one
  - SESSION\_KEY\_RCV\_STATUS = 4 Receiver: No Data received (time-out) or Data without Session Key received
  - SESSION\_KEY\_RCV\_STATUS = 0 Receiver: The received session key do match with the expected one

## Principle

In secure mode (aka session mode), when a *sender* wants to send data to a *receiver*, it has enter in session with its peer by first requesting a one-time token (aka session key). The receiver will then be ready to receive data message containing the session key. This key is made unique for each session, and one session allows only one data transfer. The session management is obtained by using two bits of the Control Byte (see RFM69 Library), the SESSION\_KEY\_REQUESTED and the SESSION\_INCLUDED bits.

With this mechanism, a hacker having recorded one transfer session will have few chances to reproduce identical transfer by sending endlessly the fake session request and/or data transfer. The chances of success are related to chances the receiver has a session key, which is identical to the one that has being recorded.



# RFM69 Session Key Library

- The Sender sends a frame of 0 data Bytes to the receiver with the Session Key Request bit of the control byte set and starts a watchdog timer (the wait time).
- The Receiver looping on `RFM69_SessionKey::recievedDone()` function fetches the sender frame seeing the Session Key Request generates and stores a Session Key of x Bytes (originally a 1 Byte random value , and extended to 4 Bytes as a copy of the current system time)
- The Receiver then sends to the Sender a frame of x Session Key Bytes with the Session Key Request and Session Key Included bits of the control Byte set
- The Sender then sends a data frame with the ACK Request bit of the control byte set (resulting of the initial `setWithRetry` command) together with the Session Included bit set followed by the Session Key
- The Receiver looping on `RFM69_SessionKey::recievedDone()` detects a data frame with the Session Included bit, verifies that the included Session Key matches the last one saved.  
If this is the case, the data is successfully stored in the RFM Data Buffer

Notes:

1. No Check is done on the Sender ID
2. The sender as no flag indicating that a session is in progress, and is therefore not controlled by a watchdog timer

- In order to complete the transaction, the Receiver has to check the ACK Request bit through the `RFM69::ACKRequested()` function
- The Receiver replies with a `RFM69_SessionKey::sendACK()` function having the control bits SendACK and Session Included

*Note: With this current version, the sendACK function is optionally repeated 2 times during the wait time (in total 3 times), ensuring that at least one Acknowledgement is received by the sender, finalizing both send and receive functions positively.*

## RF Frame

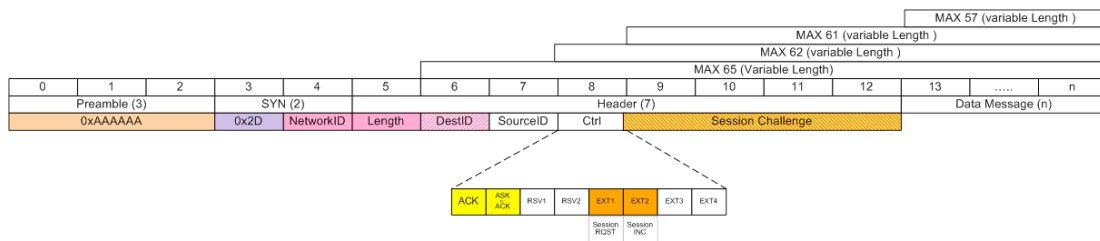


Figure 2: RF frame with session key header

- Extension Bytes 1 and 2 are used for Session Key Request and Session Key Included
- Actual size of Data payload is 57 Bytes

## Erroneous cases

### Session Key doesn't match

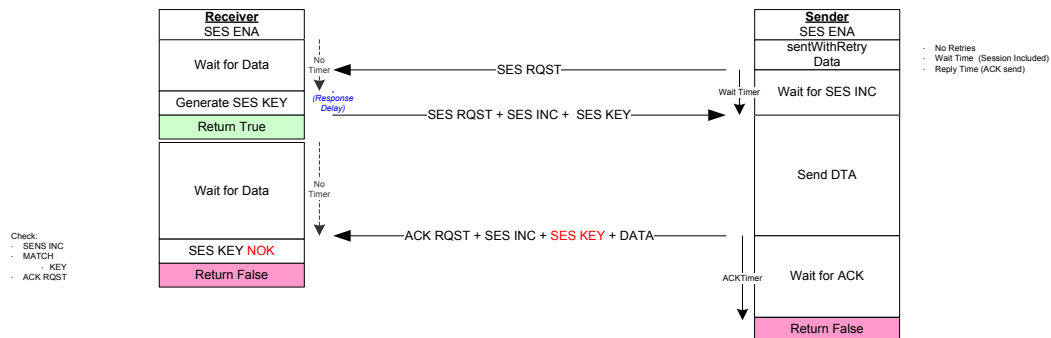


Figure 3: Session key doesn't match

- In this case the Sender function will times out returning a false status, while the receiver will return a false status because the key doesn't match

### No Session Key response from the Receiver

# RFM69 Session Key Library

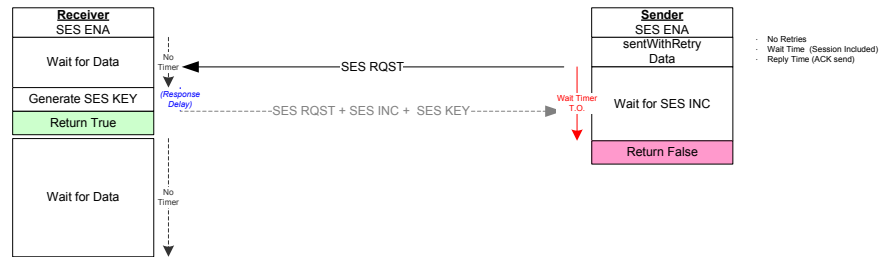


Figure 4: Sender doesn't receive the Session Key

- In this case the Sender function will times out returning a false status, while the receiver will "endlessly" wait for a Data packet with a session Key

Notes:

- This is not an issue because if the receiver receives a data packet coming from another node the Control Byte and the Session Key will not match
- If another node tries to initiate a session, a new session key will be computed

## No Data received

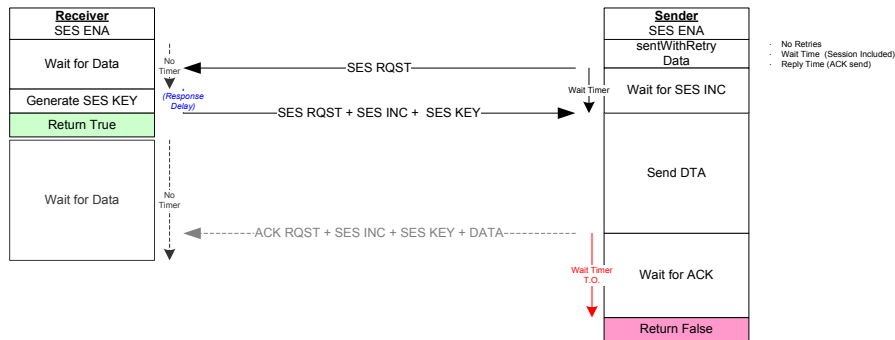


Figure 5: The Receiver doesn't receive the data

- This has an identical result as the previous case.

## No ACK received by the Sender

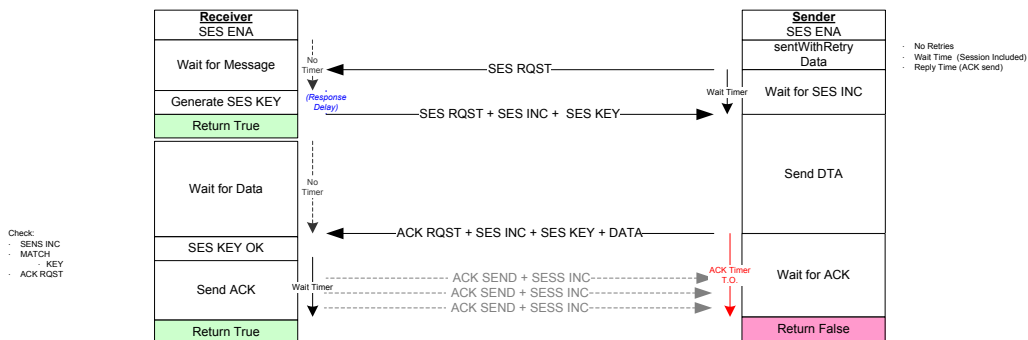


Figure 6: The Sender doesn't receive the Acknowledgement

- In this case the Sender function will times out returning a false status, while the receiver will successfully compute the data.

Notes:

- There is no simple solution for that issue
- This situation may be improved by multiplying the acknowledgement during the watchdog window, expecting that at least one ACK will be received
- It can be up to the application software to repeat the data message until an acknowledgement is received (using or not a frame counter to control duplicate data)
- An alternative; if no acknowledgement is required, is to use the `RFM69_SessionKey::send()` function instead of the `RFM69_SessionKey::sendWithRetry()` one, this will also speed-up the transaction. (see figure below), however asymmetrical results may occurs is the session key or the control Byte doesn't match.

# RFM69 Session Key Library

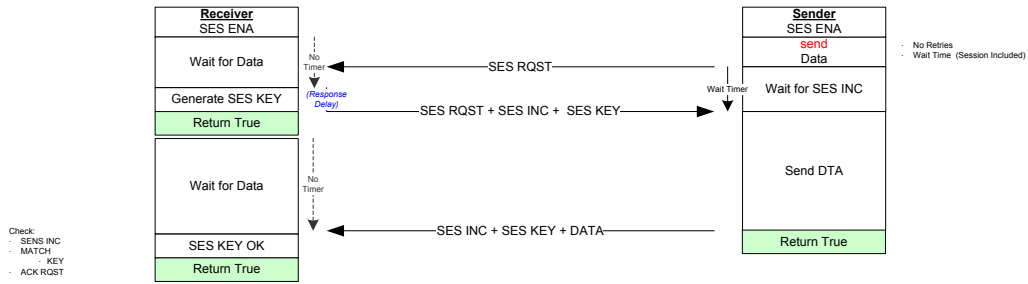


Figure 7: Datagram mode no ACK Requested

- The issue with this option is that if the Sender sends a wrong key, it will not be informed about the failure.

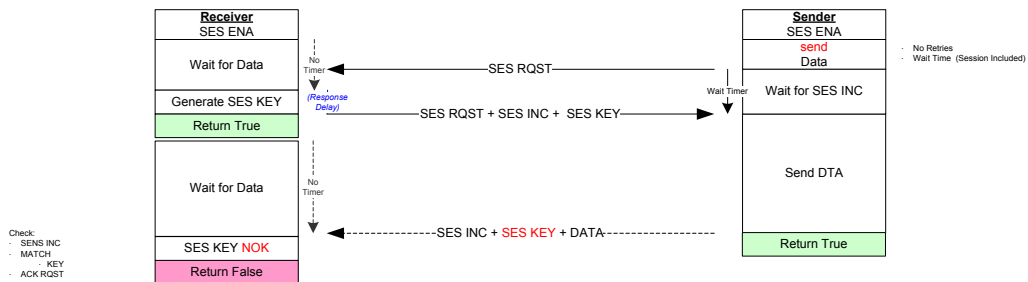


Figure 8: Datagram mode no ACK Requested sender uses a wrong key

## RFM69 SessionKey library<sup>1</sup>

*Note: The following describes the modified version of RVDB of August 2016*

### RFM69 SessionKey variables and constants

#### Configuration Constants

- SESSION\_KEY\_LENGTH  
The session key length: value is 4
- RF69\_HEADER\_LENGTH  
RFM standard header length: value is 4
- SESSION\_HEADER\_LENGTH  
The session header length:  
RF69\_HEADER\_LENGTH + SESSION\_KEY\_LENGTH
- SESSION\_MAX\_DATA\_LEN  
The Session maximum data length:  
RF69\_MAX\_DATA\_LEN - SESSION\_KEY\_LENGTH

#### Working Variables (public)

- SESSION\_KEY\_REQUESTED  
0 when false, 0x08 when true
- SESSION\_KEY\_INCLUDED  
0 when false, 0x04 when true
- SESSION\_KEY\_RCV\_STATUS

### Constructor

## RFM69\_SessionKey

### Description

A call to RFM69\_SessionKey creates an RFM69\_SessionKey object, whose name needs to be provided while calling the class.

### Notes

<sup>1</sup> See [https://github.com/dewoodruff/RFM69\\_SessionKey](https://github.com/dewoodruff/RFM69_SessionKey)

# RFM69 Session Key Library

---

The RFM69\_SessionKey inherit of all the RFM69 functions

## Syntax

```
RFM69_SessionKey();
```

```
RFM69_SessionKey(uint8_t slaveSelectPin=RF69_SPI_CS, uint8_t interruptPin=RF69_IRQ_PIN, bool  
isRFM69HW=false, uint8_t interruptNum=RF69_IRQ_NUM)
```

```
RFM69(slaveSelectPin, interruptPin, isRFM69HW, interruptNum)
```

## Parameters

- *none:*  
Default slaveSelectPin is RF69\_SPI\_CS, default interruptPin is RF69\_IRQ\_PIN, default isRFM69HW is false (RFM69W), default interruptNum is RF69\_IRQ\_NUM.
- *slaveSelectPin:*  
Used to select an SS pin. Default is determined by the hardware,
- *interruptPin:*  
Used to select an interrupt pin. By default defined by RFM69.h, i.e. 2 for ATmega328P, ATmega644P ATmega1284P
- *isRFM69HW:*  
Used to indicate the type of RFM69 module (normal RFM69W or high power RFM69HW). Default is false (i.e. RFM69W)
- *interruptNum:*  
Used to select the interrupt number. The default from RFM69.h is 0

## Example

```
RFM69_sessionKey.radio;  
RFM69_sessionKey.radio(10,2,false,0); // Setup one RFM69W instance with SS on pin 10 and an IRQ pin 2, IRQ number  
0
```

## RFM69 SessionKey Functions

### Syntax

```
RFM69_SessionKey.initialize  
RFM69_SessionKey.useSessionKey  
RFM69_SessionKey.sessionKeyEnabled  
RFM69_SessionKey.useSession3Acks  
RFM69_SessionKey.session3AcksEnabled  
RFM69_SessionKey.sessionWaitTime  
RFM69_SessionKey.sessionRespDelayTime  
RFM69_SessionKey.send  
RFM69_SessionKey.sendACK  
RFM69_SessionKey.receiveDone
```

## RFM69\_SessionKey.initialize

### Description

This public function initializes the RFM69 registers and returns a Boolean status “true” when terminated. This function must be invoked before any attempt to send or receive data.

### Syntax

```
RFM69_sessionKey.initialize(uint8_t freqBand, uint8_t ID, uint8_t networked=1)
```

### Parameters

- *freqBand:*  
Frequency band to use, this one should match the module specification. Possible values are:
  - RF69\_433MHZ
  - RF69\_868MHZ
  - RF69\_915MHZ
- *ID:*
  - The Node Identification number. Possible values are from 0 to 254 with 255 as the broadcast node ID.

# RFM69 Session Key Library

---

- *networkID*:
  - The Network Identification number. Possible values are from 0 to 255, default is 1

## Returns

ALWAYS true when initialization is terminated

## Notes

1. This function initializes the following parameters before calling the standard RFM69 initialization:
  - `_sessionKeyEnabled` default session negotiation set to disabled
  - `SESSION_KEY_INCLUDED` default Control Byte option set to 0
  - `SESSION_KEY_REQUESTED` default Control Byte option set to 0
  - `_waitTime` default wait time between Session Request and Session included is 40 ms
  - `_respDelayTime` default response delay time between a SESSION KEY request and the RF response (used for slow responsive node). Default value is 0  $\mu$ s
2. Encryption is disabled after initialization.
3. The node ID is not saved into the NodeAddress Register (RegNodeAdrs), because address filtering is by default not used.

## Example

```
#define NODEID      99
#define NETWORKID   100
#define FREQUENCY   RF69_433MHZ
...
radio.initialize(FREQUENCY, NODEID, NETWORKID);
```

## RFM69\_SessionKey.useSessionKey

### Description

Set or reset the internal `_sessionKeyEnabled` variable, allowing activating or not the session key negotiation.

### Syntax

RFM69\_SessionKey.useSessionKey(bool enabled);

### Parameters

- *enabled*:
  - The Session Key option. Possible values are 0, false (disabled) or 1, true (enabled), default is disabled

### Returns

None

### See also

RFM69\_SessionKey.useSessionKey

### Example

```
radio.useSessionKey(true);
```

## RFM69\_SessionKey.sessionKeyEnabled

### Description

Boolean function that returns the status of the Session Key option

### Syntax

RFM69\_SessionKey.sessionKeyEnabled()

### Parameters

None

### Returns

True if Session Key is enabled

### See also

RFM69\_SessionKey.useSessionKey

### Example

```
if (radio.sessionKeyEnabled()) Serial.println ("Session Key is Enabled");
else Serial.println ("Session Key is Disabled");
```

# RFM69 Session Key Library

---

## RFM69\_SessionKey.useSession3Acks

### Description

Set or reset the internal *\_session3AcksEnabled* variable, allowing to activate or not the final 3 Acks . This option enforces 3 final Acks instead of 1 to cope with possible the lost of final acknowledgement leaving Sender and Receiver with asymmetrical results

### Syntax

```
RFM69_SessionKey.useSession3Acks(bool enabled);
```

### Parameters

- *enabled*:
  - The Session final 3Acks option. The possible values are 0, false (disabled) or 1, true (enabled). Default is disabled.

### Returns

None

### See also

RFM69\_SessionKey. session3AcksEnabled

### Example

```
radio.useSession3Acks(true);
```

## RFM69\_SessionKey. session3AcksEnabled

### Description

Boolean function that returns the status of the Session 3 Acks option

### Syntax

```
RFM69_SessionKey.session3AcksEnabled ()
```

### Parameters

None

### Returns

True if Session 3 Acks is enabled

### See also

RFM69\_SessionKey. useSession3Acks

### Example

```
if (radio.session3AcksEnabled()) Serial.println ("Session 3 Acks Enabled");  
else Serial.println ("Session 3 Acks is Disabled");
```

## RFM69\_SessionKey. sessionWaitTime

### Description

This function allows changing the watchdog time between the Session Request and the Session Included response.

### Syntax

```
RFM69_SessionKey.sessionWaitTime(uint8_t waitTime);
```

### Parameters

- *waitTime*:  
The maximum time in ms between the moment the sender sends a Session Request and the time the receiver answers with a Session Key and a Session Included control Bytes, default is 40ms

### Returns

None

### Note



# RFM69 Session Key Library

---

By default the wait time is set identical to the default value of the *RFM69.sendWithRetry* function, however changing the retry wait time parameter of the *RFM69.sendWithRetry* function, do not affect the Session wait time. This is the reason why this function was added.

## See also

RFM69. sendWithRetry

## Example

```
radio.sessionWaitTime(70); // Set the session wait time to 70ms
```

## RFM69\_SessionKey. sessionRespDelayTime

### Description

This function allows changing the delay between a Session Key request and the RF challenge response for slow node (typically for low power node running at a CPU slow speed, for instance 4MHz)

### Syntax

RFM69\_SessionKey.sessionRespDelayTime(uint8\_t respDelayTime)

### Parameters

- *respDelayTime*:  
The delay time in  $\mu\text{s}$  between the moment the receiver gets a Session Request and the receiver answers with a Session Key and a Session Included control Bytes. The default is  $0\mu\text{s}$ , and the maximum value is  $500\mu\text{s}$ .

### Returns

None

### Example

```
radio.sessionRespDelayTime(150); // Set the session response delay time to 150 $\mu\text{s}$ 
```

## RFM69\_SessionKey.send

### Description

This function replaces the *RFM69.send* function, it checks for a clear channel (*canSend*) for duration less than RF69\_CSMA\_LIMIT\_MS, before attempting to send (for a duration that should not exceed RF69\_TX\_LIMIT\_MS) the contents of a data buffer (length is restricted to the SESSION\_MAX\_DATA\_LEN). It essentially activates the internal *sendWithSession* function that takes care of the Session key negotiation. Setting the acknowledgement request bit of the control Byte is optional.

### Syntax

RFM69\_SessionKey.send(uint8\_t toAddress, const void\* buffer, uint8\_t bufferSize, bool requestACK=false)

### Parameters

- *toAddress*:  
The destination node ID
- *\*buffer*:  
A pointer to the data buffer
- *bufferSize*:  
The size of the relevant data part of the buffer. Data will be truncated to the RF69\_MAX\_DATA\_LEN.
- *requestACK*:  
To enforce the acknowledge request bit of the control Byte, default is none

### Returns

None

### Notes

Using *RFM69\_SessionKey.send* with the requestACK bit of the Control Byte true has the same effect than using the *RFM69.sendWithRetry* without retries, however this is not a Boolean function for which a successful transmission can't be tested.

# RFM69 Session Key Library

---

## See also

RFM69.sendWithRetry

## Example

```
radio.send(10, "Hello", 5); // Send 5 Bytes of the string "Hello" to node 10 without
                             ACK request (if the Session Key is enable, a session key negotiation will be started
                             before data are sent).
```

## RFM69\_SessionKey.sendACK

### Description

A variant of the *send* function that sends to the last received source node ID a data packet the ACK bit and the SESSION\_INCLUDED (if the Session Key is enabled) bit of the Control Byte.

### Syntax

```
RFM69_SessionKey.sendACK(const void* buffer = "", uint8_t bufferSize=0)
```

### Parameters

- *buffer*:  
A pointer to the acknowledgement data buffer. By default data is empty
- *bufferSize*:  
The size of the relevant data part of the buffer. Data will be truncated to the RF69\_MAX\_DATA\_LEN. By default the size is set to 0.

### Returns

None

## See also

RFM69\_SessionKey.send

RFM69.ACKReceived

## Notes

The original RSSI value of the last received frame is preserved, by this function.

The acknowledge datagram as by default no data; however an ACK datagram may carry data as a normal send function.

## Example

```
if (radio.ACKRequested())
{
    radio.sendACK();
    Serial.println(" - ACK sent");
}
```

## RFM69\_SessionKey.receiveDone

### Description

This Boolean function sets the RFM69 in receiving mode, allowing RFM69 reception interrupt.

Once interrupt is usefully terminated (on correctly formatted frame with data >0) data of the received frames are accessible through the working variables.

If the Session Key is enabled, only frames > 0 Bytes (not only control ones) with incoming Session Key corresponding to the stored Session Key are validated.

### Syntax

```
RFM69_SessionKey.receiveDone()
```

### Parameters

None

### Returns

True if data is received, else false.

When true the following working variables are updated.

#### **RFM69**

- DATALEN
- SENDERID
- TARGETID

# RFM69 Session Key Library

---

- PAYLOADLEN
- ACK\_REQUESTED
- ACK\_RECEIVED
- RSSI

## RFM69\_SessionKey

- SESSION\_KEY\_REQUESTED
- SESSION\_KEY\_INCLUDED

## Example

```
if (radio.receiveDone())
{
  Serial.print('[');Serial.print(radio.SENDERID, DEC);Serial.print("] ");
  Serial.print(" [RX_RSSI:");Serial.print(radio.readRSSI());Serial.print("]");
  Serial.print(" to [");Serial.print(radio.TARGETID, DEC);Serial.print("] ");
  Serial.print(" ACK Requested: "), Serial.print(radio.ACKRequested());
  Serial.print(" Data Length: "), Serial.print (radio.DATALEN, DEC);
  Serial.print(" Data: "); for (byte i = 0; i < radio.DATALEN; i++)
Serial.print((char)radio.DATA[i]);
  Serial.println();
}
```

## See also

RFM69.ACKReceived()

# RFM69 Session Key Library

---

## Appendix

### Usage

When the session key library is used, the sender shall call the *sendWithRetry* function.

This function takes care of the overall retries (default 2) and watchdog timer (default 40ms). Practically we recommend calling this function with no retries (SEND\_RTRY, = 0) and a watchdog timer or 50ms (SEND\_WAIT\_WDG = 50).

E.G:

```
radio.sendWithRetry  
(REMOTEID, (constvoid*) (&theData), sizeof(theData), SEND_RTRY, SEND_WAIT_WDG)
```

### Return Status

While requesting conditionally *sendWithRetry* data transfer with the session key enabled, extra return statuses may be obtained by retrieving the value of the public variable SESSION\_KEY\_RCV\_STATUS.

The current value are:

- 1 Receiver: A session key is requested and sent
- 2 Sender: A session key is received and computed
- 3 Receiver: The received session key doesn't match with the expected one
- 4 Receiver: No Data received (time-out) or Data without Session Key received
- 0 Receiver: The received session key does match with the expected one

Note that the actual return status of the *sendWithRetry* is geared by the RFM69 Boolean function, which is controlled by a number of retries and watchdog timer. So it might be possible that this session ended by a timeout while the session key exchange was successful (see below timer recommendations).

### Timers recommendations

Several timers are foreseen.

- **retryWaitTime:** this is the main one defined while calling the RFM69 *sendWithRetry* function, the default value is 40ms, the maximum value should be 255.

---

*Note: Typically for the longest frame (71 Bytes) round-trip time at the default RFM69 baud rate (55.55Kbps) is about 10ms (see RFM69 library).*

---

- **respDelayTime:** this one is managed by the RFM69\_SessionKey *sessionRespDelayTime* function, this is the watchdog timer that protects the reception of the session key. By default this one is also 40ms but should be set to a lower value than the previous one
- **sessionWaitTime:** this one is managed by the RFM69\_SessionKey *sessionWaitTime* function, it is used to slow down the response to a session key request for slow nodes (typically nodes configured with a slow clock rate and sleep mode). By default this value is 0 with a maximum of 500µs
- **useSession3Acks:** a RFM69\_SessionKey function that may be used by the receiver node to improve the reception of the final ACK by the sender. By activating this function, 2 extra ACK frames are generated at the end of the transaction with a delay equal to 1/4 of the *respDelayTime* (or by default 10ms). The 3 ACKs option are by default disabled.

So for the time being the timing recommendation are:

retryWaitTime set to 50ms - the session key exchange may be a little bit longer than a normal transfer, (however 40ms gives already sufficient margin).

# RFM69 Session Key Library

---

respDelayTime: 8ms , the session key is only 4 Bytes, so the watchdog timer should be shorter than the one of a data frame.

It ensures that even with a 3 ACKS a transfer should occur in the frame of the overall retryWaitTime

## *How does it works*

### New Control Bytes

An RFM69 frame includes one Control Byte, from which two bits are currently used to indicate that a frame requests an acknowledgement or is an acknowledgement:

- RFM69\_CTL\_SENDACK 0x80
- RFM69\_CTL\_REQACK 0x40

The usage of the RFM69\_SessionKey library requires the definition of two new bits of the Control Byte to indicate that a Session Request transfer is required and that the Data transferred includes a Session Key.

- RFM69\_CTL\_EXT1 0x08 // RFM\_SessionKey: Session requested
- RFM69\_CTL\_EXT2 0x04 // RFM\_SessionKey: Session included

### Interrupt Handler Hook

The interrupt handler hook of the RFM69 library allows when an RFM69 interrupts occurs to extend the management to the RFM69\_SessionKey through extra check of the Control Byte this take appropriate actions in the frame of the Session Key management.

### Virtualization

Virtualization of library functions allows bypassing the normal usage of a function by an identical defined in another library.

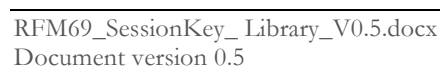
The RFM69\_SessionKey library uses the virtualization of some RFM69 functions such as:

- bool **initialize**(uint8\_t freqBand, uint8\_t ID, uint8\_t networkID=1);  
The corresponding RFM69\_SessionKey function as the same function as the RFM69 one, and initializes some extra RFM69\_SessionKey extra parameters
- void **send**(uint8\_t toAddress, const void\* buffer, uint8\_t bufferSize, bool requestACK=false);  
This function checks if the frame to send is to be one requesting a session or not, if it doesn't requires a session, then the native RFM69 *SendFrame* function is called
- void **sendACK**(const void\* buffer = "", uint8\_t bufferSize=0);  
In case a session is enabled, this function bypasses the RFM69 one by updating the control byte with the session key extensions bits. If no session specific acknowledgement is required the RFM69 function is called.
- bool **receiveDone**();
- In case of session, an extra check is done against the received frame to verify if session key matches the generated one.

### Session Key enabled (true)

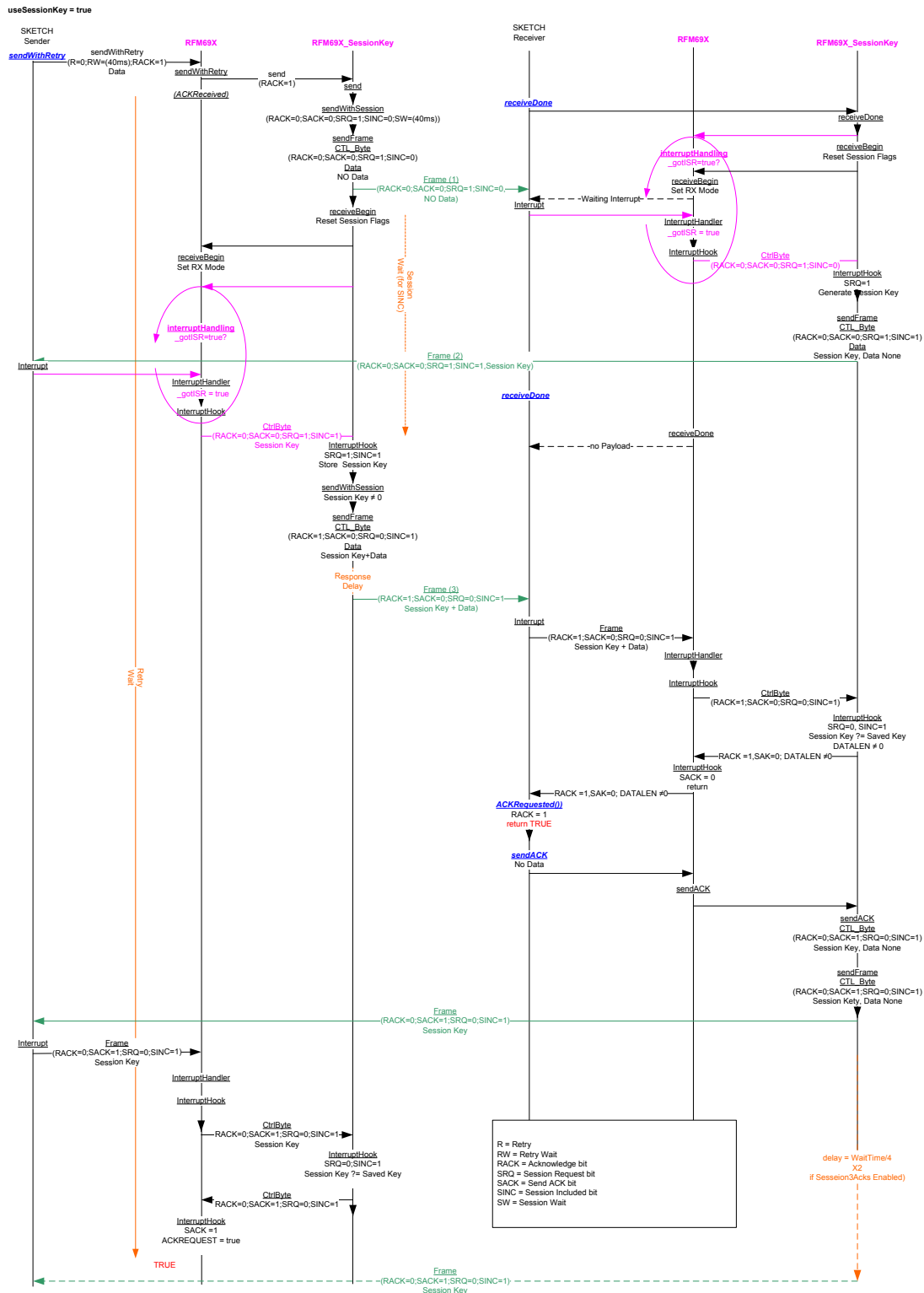
The following diagrams summarize the transfer between a sender and a receiver while Session management is enabled at both sides.

## RFM69\_SessionKey for AVR and ESP8266



# RFM69 Session Key Library

## RFM69\_SessionKey for ES32



## Session Key tracing

For a better understanding, the following traces is provided

## Legend:

- 1 represents RFM69 functions
  - 2 represent RFM69\_SessionKey functions
- SENDER**

# RFM69 Session Key Library

---

```
2 - sessionKeyEnabled() = true
1 - sendWithRetry(GATEWAYID, payload, sendSize, 0) // Will NOT initiate Retries and Default wait time = 40ms
2 - send(GATEWAYID, payload, sendSize, true) // RQSTACK
    if (sessionKeyEnabled()) // True > 2 - sendWithSession
2 - sendWithSession(GATEWAYID, payload, sendSize, true) // RQSTACK
    SESSIONKEY = 0
2 - sendFrame(GATEWAYID, null, 0, false, false, true, false) // NO Data, NO RQTTACK, NO
SENDACK, SESRQST NOSESSINC
    FIFO SPI Transfer 4 Bytes Header with CTRLByte = SESSRQST and SESSIONKEY = 0
    while (millis() - sentTime < retryWaitTime && SESSION_KEY == 0); // Check for T.O. and SESSION KEY
    = 0
2 - receiveBegin() // reset SESSION Ctl Bytes
1 - receiveBegin() // Wait for Interrupt from Receiver
    1 - interruptHandler() // Get Receiver DATA with basic test (payload length, if not promiscuous, Target ID or
Broadcast ID match)
        2 - interruptHook(CTLbyte)
            if (sessionKeyEnabled() && SESSION_KEY_REQUESTED &&
SESSION_KEY_INCLUDED)
                Check if received data has only SESSRQST and SESINC to read the Session KEY
                SPI Transfer SESSION_KEY = KEY
            1 - Return to interruptHandler
                Save data of payload if included in the ACK
        2 - Return to sendWithSession
        2 - sendFrame(GATEWAYID, payload, sendSize, true, false, false, true)
            // Send Data with RQSRACK (from initial), NO SENDACK, NO SESRQST SESSINC + SESSION_KEY
1 - Return to SendWithRetry3
    while (millis() - sentTime < retryWaitTime) // Wait now for the final ACK !!!
    {
        if (ACKReceived(GATEWAYID))
1 - ACKReceived(GATEWAYID)
    1 - receiveDone()
        1 - interruptHandler() // Get Receiver DATA with basic test (payload length, if not promiscuous, Target ID or
Broadcast ID match)
            2 - interruptHook(CTLbyte)
                if (sessionKeyEnabled() && SESSION_KEY_INCLUDED &&
!SESSION_KEY_REQUESTED)
                    Check for data with SESINC only + Check if INCOMING_SESSION_KEY = SESSION_KEY
                1 - return to interruptHandler
                    Save data of payload if included and update the ACK CTRLByte
            1 - Return to ACKReceived()
                True if ACKRCV and Sender ID or Broadcast ID match
1 - Return true
```

## RECEIVER

```
2 - receiveDone() // Wait for Interrupt from Sender (Session Request for instance)
2 - receiveBegin() // reset SESSION Ctl Bytes
1 - receiveBegin()
    1 - interruptHandler() // Get Sender DATA with basic test (payload length, if not promiscuous, Target ID or Broadcast ID
match)
        2 - interruptHook(CTLbyte)
            if (sessionKeyEnabled() && SESSION_KEY_REQUESTED &&
!SESSION_KEY_INCLUDED)
                Check if received data has SESSRQST
                > Generate Session KEY
            2 - sendFrame (SENDERID, null, 0, false, false, true, true) // NO Data, NO
RQTTACK, NO SENDACK, SESRQST SESSINC
                FIFO SPI Transfer 4 Bytes Header with CTRLByte = SESSRQST and SESSION_KEY = KEY
            2 - Return
            1 - Return
        1 - Return
1 - Return (true)
2 - receiveDone() // Wait for Interrupt from Sender Data with Session for instance
```

---

<sup>2</sup> No check done on the actual size of the payload which should be max RF69\_MAX\_DATA\_LEN-1

<sup>3</sup> If Retry was specified in the initial command, it will start here



# RFM69 Session Key Library

---

```
2 - receiveBegin () // reset SESSION Ctrl Bytes
1 - receiveBeign()
1 - interruptHandler() // Get DATA with basic test (payload length, if not promiscuous, Target ID or Broadcast ID match)
2 - interruptHook(CTLbyte)
    if (sessionKeyEnabled() && SESSION_KEY_INCLUDED &&
    !SESSION_KEY_REQUESTED)
        Check for data with SESINC only + Check if INCOMING_SESSION_KEY = SESSION_KEY
1 - return to interruptHandler
    Save data of payload if included and update the ACK CTRL.Byte
2 - Return
1- Return
1 - Return
1 - Return (true)

1 - if (ACKRequested()) // Necessary to fetch the sendWithSession final ACK Request
    Check the CTLByte for ACK_REQUESTED from any Node address except Broadcasts
2. - sendACK() // With NO Data
    1 - sendFrame(sender, buffer, bufferSize, false, true) // With No Data NO RQSTACK but with SENDACK
1 - Return
```

-ooOoo-