

# RFM69 Session Key Library

---

Library Version: February 2015  
Library Reference: [https://github.com/dewoodruff/RFM69\\_SessionKey](https://github.com/dewoodruff/RFM69_SessionKey)  
Document Version: 0.0  
Save Date: 26 October 2015

## 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.

---

*Note: The RFM69 library of 2<sup>nd</sup> of October 2015, include the function virtualization, however, the extension of the control Bytes used by the Session Key library need to be added (see Appendix: Control Byte extension (RFM69.h)).*

---

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. Optionally 3 ACKs are sent by the receiver at the end of the session instead of one (a best effort attempt to solve undelivered ACK message)
6. New function (setWaitTime) is added, allowing changing the watchdog time between a Session Request and a Session Include handshake, this function uses \_waitTime as private variable.

## 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.

The security of the data transfer is geared by two principles;

1. A hacker can't guess the session key generation algorithm, to anticipate the one a receiver will expects
2. The algorithm that generates the key has a long or complex rollover mechanism

For the first point it is obvious that the transaction is to be encrypted.

For the second point, the original library uses a pseudorandom generated Byte, which we believe has a too short rollover scheme, therefore the session key is modified to be a 4 Bytes as the system uptime snapshot. The system time as a rollover period (if no intermediate reboot) of approximately 59 days.

## Working cases

1. Both Sender and Receive nodes have session key disabled: There is no session key negotiation, this is the default RFM69 mode
2. Both Sender and Receive nodes have session key enabled. This is the secure transaction mode, see description below
3. Sender disabled, Receiver enabled. There is no session key negotiation, allowing "fire and forget" messages from nodes that don't require session keys.

# RFM69 Session Key Library

- Sender enabled, Receiver disabled – This configuration doesn't work, because the receiver can't handshake a key.

## Evaluation

### Pros

- Provides reasonable protection against replay attacks. Helpful for sensitive transactions such as opening a lock or garage door
- It is flexible by the usage of session Key option flag that can be turned on and off according to the capability of the receiver node.

### Cons

- While the sender waits for a response it can't receive data during a period controlled by a wait timer. There is the potential of miss connections from other nodes trying to access the sender during this time, which could be a problem on busy networks.
- Data transaction in Session mode takes longer than a normal transaction; 3 to 4 (if ACK requested) exchanges vs. 1 to 2 (if ACK Requested) exchanges.
- Size of the Data is reduced from 61 to 57 Bytes

## Examples

### Secure transaction

#### Sender and Receiver have the Session enable option

The following diagram summarizes a normal session key transaction.

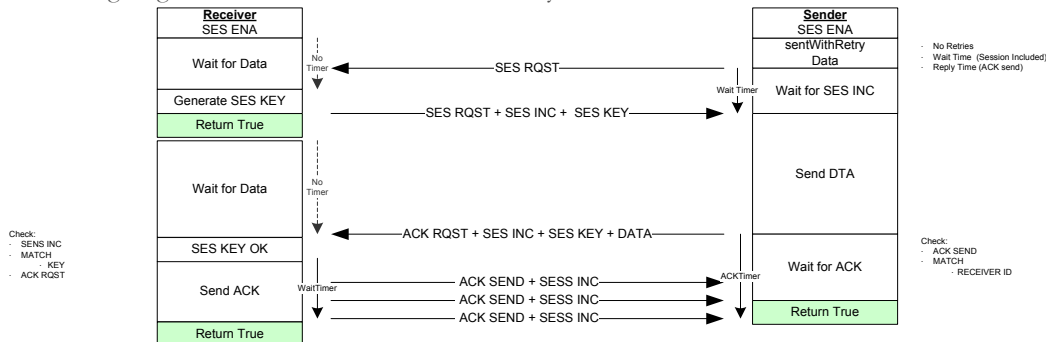


Figure 1: RFM Session Key transaction

Both Sender and Receiver are session enable (`RFM69_SessionKey::useSessionKey(1)`)

The sender starts the session through the following function:

`RFM69_SessionKey::sendWithRetry(Receiver, payload, sendSize, 0)`, meaning no retries with default wait time of 40ms.

- 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::receivedDone()` 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::receivedDone()` 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:

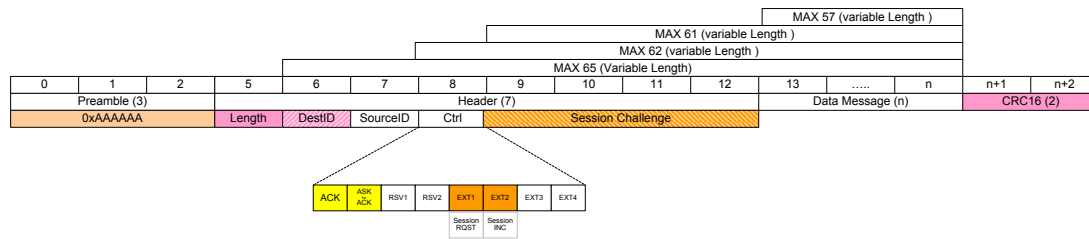
# RFM69 Session Key Library

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.*

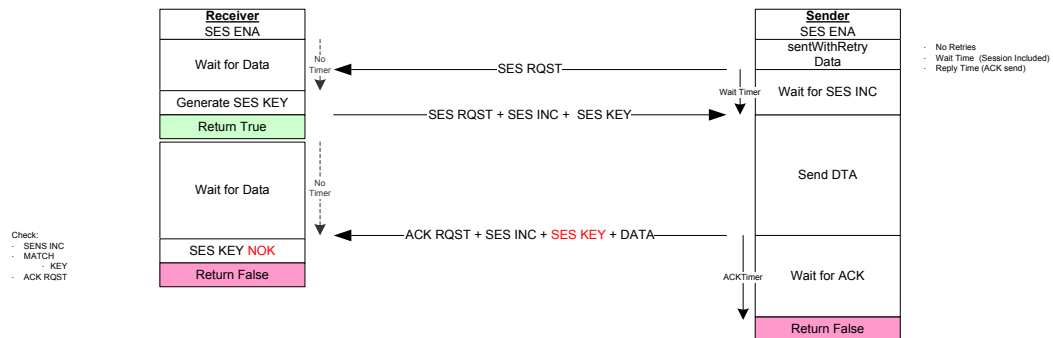
## RF Frame



- 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



- 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

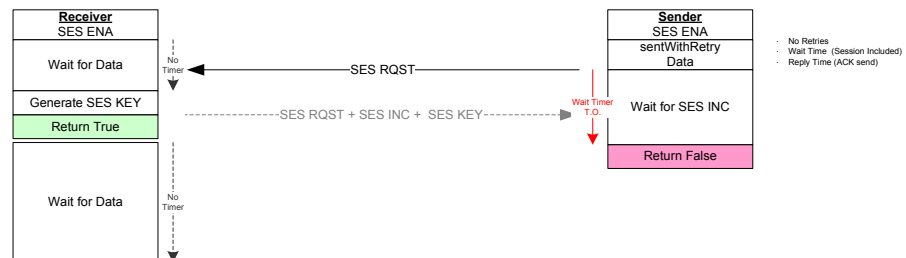


Figure 2: Sender doesn't receive the Session Key

- 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:

1. 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
2. If another node tries to initiate a session, a new session key will be computed

# RFM69 Session Key Library

## No Data received

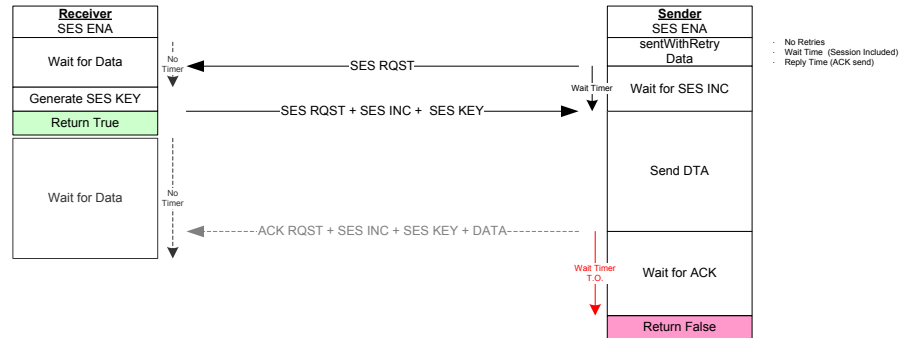


Figure 3: The Receiver doesn't receive the data

- This has an identical result as the previous case.

## No ACK received by the Sender

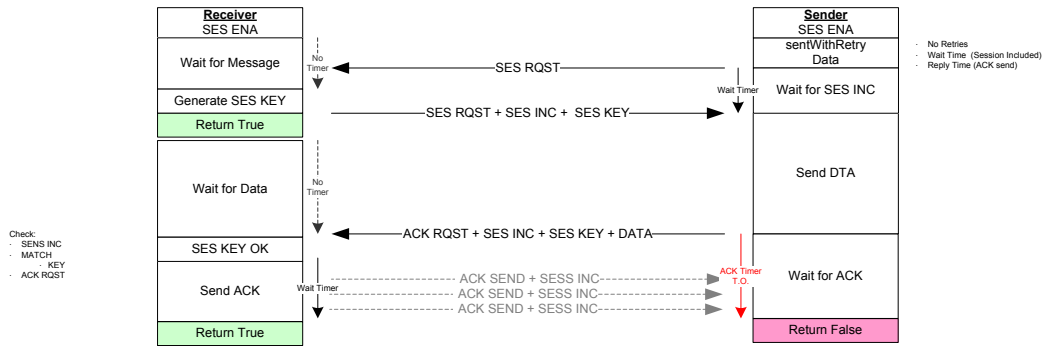


Figure 4: The Sender doesn't receive the Acknowledgement

- 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.

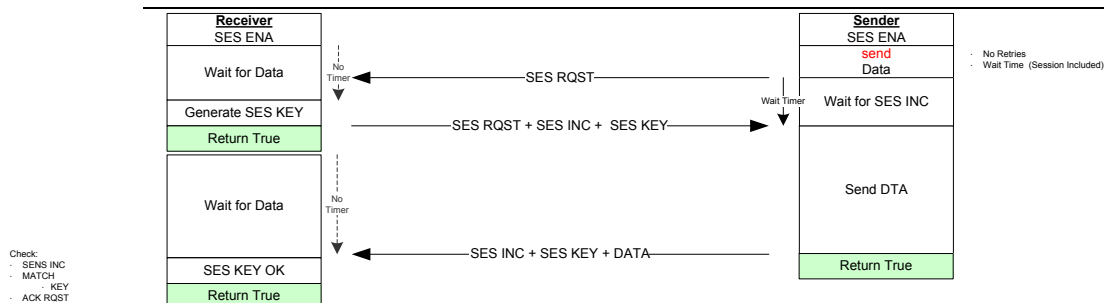


Figure 5: Datagram mode no ACK Requested

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

# RFM69 Session Key Library

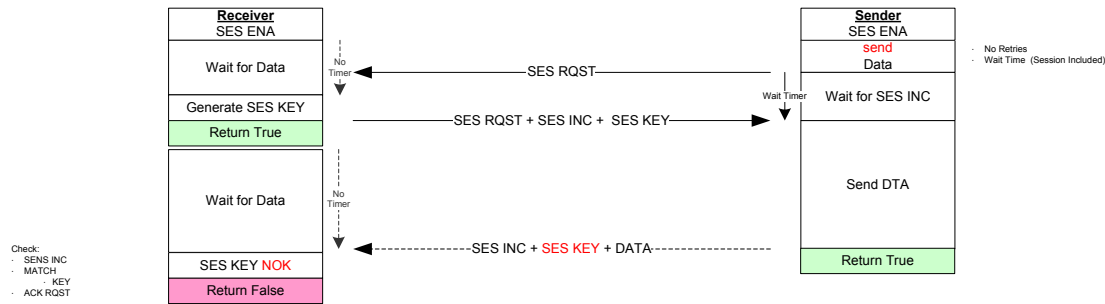


Figure 6: Datagram mode no ACK Requested sender uses a wrong key

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

*Note: The following describes the modified version of RVDB of July 2015*

### 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

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

### 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

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.

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

# RFM69 Session Key Library

---

- *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.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. 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
```

# RFM69 Session Key Library

---

## Example

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

```
RFM69_SessionKey.sessionWaitTime
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.
- *networkedID*:
  - 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
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 to activate or not the session key negotiation.

### Syntax

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

### Parameters

# RFM69 Session Key Library

---

- *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\_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. 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 Session Key Library

---

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

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.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

# RFM69 Session Key Library

---

- *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.

## 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.

# RFM69 Session Key Library

---

## 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
- 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

### Control Byte extension (RFM69.h)

```
// TWS: define CTLbyte bits
#define RFM69_CTL_SENDACK 0x80
#define RFM69_CTL_REQACK 0x40
//!!!
#define RFM69_CTL_RESERVE1 0x20
#define RFM69_CTL_RESERVE2 0x10
#define RFM69_CTL_EXT1 0x08 // reserve for RFM69 derived classes to use
#define RFM69_CTL_EXT2 0x04
#define RFM69_CTL_EXT3 0x02
#define RFM69_CTL_EXT4 0x01
```

### Session Key tracing

#### Legend:

1 represent RFM69 functions

2 represent RFM69\_SessionKey functions

#### SENDER

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 = **SESRQST** 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 **SESRQST** 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<sup>2</sup>, **true**, false, false, **true**)

// Send Data with **RQSTACK** (from initial), NO SENDACK, NO **SESRQST** **SESSINC** + **SESSION\_KEY**

1 - Return to SendWithRetry<sup>3</sup>

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)

---

<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

---

```

    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 ACKReceicved()
    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 - receiveBeign()
            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 RQSTACK, NO SENDACK, SESSRQST 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)
    2 - receiveBegin () // reset SESSION Ctl 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 CTRLByte
            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-