CC2420 Security Tutorial

From TinyOS Wiki

This tutorial explains how the CC2420 in-line security features can be enabled in an application. Using the security features require modifications to the Makefile, configuration file, and the implementation file.

Note: CC2420 Security Features are part of TinyOS 2.1.1.

Contents

Introduction

The CC2420 radio chip supports three types of in-line security modes, leveraging the same underlying 128-bit AES encryption:

- Counter Mode Encryption (CTR)
- Cipher Block Chaining Message Authentication Code (CBC-MAC)
- Counter with CBC-MAC (CCM).

The CC2420 in-line security implementations add two new interfaces to the CC2420 radio stack in TinyOS 2.1: CC2420SecurityMode and CC2420Keys. The implementations are located in tos/chips/cc2420/security/ (http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/tos/chips/cc2420/security/) and the interfaces are located in tos/chips/cc2420/interfaces/ (http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/tos/chips/cc2420/interfaces/) . The design of the security implementation is based on the CC2420 specifications (http://focus.ti.com/lit/ds/symlink/cc2420.pdf) and the IEEE 802.15.4 2006 standards (http://www.ieee802.org/15/pub/TG4.html) .

Transmitter Configuration

Makefile

Users intending to enable the security features MUST add the CC2420_HW_SECURITY flag in the Makefile.

```
CFLAGS+=-DCC2420_HW_SECURITY
```

Another point to note is the possible need for modifying the value of TOSH_DATA_LENGTH. Using different security options will add different amounts of additional overhead in the packet. For example, using the CBC-MAC authentication with a 16 byte MIC will require an additional 16 bytes in the payload portion of the message_t. While the security header is located in the cc2420_header_t, it takes up 6 additional bytes in the packet as well. The format of the security header can be found in tos/chips/cc2420/CC2420.h (http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/tos/chips/cc2420/CC2420.h?view=markup)

Wiring (configuration file)

```
components new SecAMSenderC(AM_RADIO_COUNT_MSG) as AMSenderC;
components new AMReceiverC(AM_RADIO_COUNT_MSG);
components CC2420KeysC;
App.Receive -> AMReceiverC;
App.AMSend -> AMSenderC;
App.Packet -> AMSenderC;
App.CC2420SecurityMode -> AMSenderC;
App.CC2420Keys -> CC2420KeysC;
```

The AMSender interface MUST be wired to the SecAMSenderC component. The Packet interface is also provided by the SecAMSenderC and all packets that use the CC2420 in-line security features MUST be wired to this component. Note that the CC2420SecurityMode interface explained in the previous section is also provided by the SecAMSenderC component. The CC2420Key interface is provided by the CC2420KeyC component. The Receive interface can be wired as the case when no security is used because decryption happens transparently for the secured packets at the lower layers (below AM stack).

Implementation File

An array of 16 bytes SHOULD be set to store the desired key values. An example is shown below.

```
uint8_t key[16] = {0x98,0x67,0x7F,0xAF,0xD6,0xAD,0xB7,0x0C,0x59,0xE8,0xD9,0x47,0xC9,0x71,0x15,0x0F};
```

After the radio starts (SplitControl.startDone()), the following commands SHOULD be called to set the key values to a desired key register as explained above. The example below sets register 1 of the key registers (CC2420 offers registers 0 and 1) to a user specified key value shown above.

```
call CC2420Keys.setKey(1, key);
```

This call to the setKey command signals an event indicating the end of the key setting process.

```
event void CC2420Keys.setKeyDone(uint8_t keyNo, uint8_t* skey){}
```

This event is important for both the transmitting node. When this event is signaled, one of the following commands (provided by the CC2420SecurityMode interface) can be called for each packet transmission, with respect to the user-defined key values.

```
call CC2420SecurityMode.setCtr(msg, 1, 0);
call CC2420SecurityMode.setCbcMac(msg, 1, 0, 16);
call CC2420SecurityMode.setCcm(msg, 1, 0, 16);
```

For *setCbcMac* and *setCcm*, the last parameter is the size in bytes of the message authentication code. It can be 4, 8 or 16. Once the above steps are done,

```
call AMSend.send(msg, len);
```

can be called to send a packet just like any other packet transmissions.

Receiver Configuration

A receiver node that intends to enable the CC2420 Security features MUST add the CC2420_HW_SECURITY flag in the Makefile as well as the transmitter.

```
CFLAGS+=-DCC2420_HW_SECURITY
```

This enables all the decryption processes in the CC2420ReceiveP.nc file. Also, the receiver must have knowledge about the key values that a transmitter is using and SHOULD set the key registers with the user-desired keys before packets are exchanged. The example below sets register 1 of the key registers (CC2420 offers registers 0 and 1) to a user specified key value.

```
uint8_t key[16] = {0x98,0x67,0x7F,0xAF,0xD6,0xAD,0xB7,0x0C,0x59,0xE8,0xD9,0x47,0xC9,0x71,0x15,0x0F};
call CC2420Keys.setKey(1, key);
```

This call to the setKey command will signal an event indicating the end of the key setting process.

```
event void CC2420Keys.setKeyDone(uint8_t keyNo, uint8_t* skey){}
```

To use the CC2420Keys interface above, the following wiring MUST be done in the configuration file.

```
components CC2420KeysC;
App.CC2420Keys -> CC2420KeysC;
```

For the receiver, this event indicates that the radio is now ready to decrypt packets with the user-defined key values.

Examples

Sample implementations of applications that enable the CC2420 in-line security features (RadioCountToLeds and BaseStation) can be found in apps/tests/cc2420/TestSecurity/ (http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/apps/tests/cc2420/TestSecurity/) .

See also

- IEEE 802.15.4
- TinySec

External links

- Paper that include evaluation of CC2420 computational overhead (see section 4.2) (http://doi.acm.org/10.1145/1514274.1514277)
- Paper that include CC2420 CCM energy evaluation (see section IV-B) (http://www2.computer.org/portal/web/csdl/doi/10.1109/SENSORCOMM.2009.29)

Categories: CC2420 | Security | Tutorials
