OpenHR20 – Securing Wireless Communication

Authors:
2009 Dario Carluccio (hr20-at-carluccio-dot-de)
2009 Jiri Dobry (jdobry-at-centrum-dot-cz)

Changes:

30.1.2009	jdobry – allow “dummy” sync packets
13.2.2009	jdobry – fix encryption chapter
	jdobry – change Kenc and Kmac generation
19.3.2009	jdobry – endian encoding on sync packet

1 Overview

The Rondostat HR20 is a radiator thermostat from Honeywell. Aim of the OpenHR20 Project it to write a new open source firmware with more functionality for this hardware. The project is hosted at sourceforge.net [sf]

A part of this project is to equip the HR20 hardware with additional wireless transceivers, e.g.: RFM12 Modules [rfm], but also other data transfer interfaces.

Especially wireless interfaces have the disadvantage, that it can not by controlled who is able to receive the data which are transmitted. Also it is not distinguishable from the received data if it was send by the system or by a third (unauthorized) entity.

For this reasons it must be defined how mechanisms of it-security can be used to secure the communication in the system.

2 Topology

The system contains one central master device and at least one slave device. The slave devices are the HR20 thermostats. The master is a dedicated hardware which is able to receive and send data from and to the slaves.

As the slaves runs on battery it is not possible to keep the receiver active all the time. Each data transfer is initiated by the slave in a defined interval (e.g.: every 60 seconds). After a packed was send from the slave it keeps the receiver a short time frame active, so that the master is able to send data to the thermostat.
Note: media access is handled by other measures, we do not have to care about this here.

3 Requirements

Each slave has a unique “ID”, which is used to identify a slave in the network.
We detonate “data” as the data which is transmitted between master and slave. Data can be status from the thermostat as well as commands from the master to the slave.

- **R1: An unauthorized entity must no be able to read any “data”**
  It is not convenient if a burglar can distinguish if someone is at home by reading the actual room temperature from the outside.
- **R2: An unauthorized entity must no be able to write any “data”**
  Nobody else must be able to control the behavior of the thermostat. Also nobody must not have the possibility to act as a slave and send fake data to the master.
- **R3: Replay must not be possible**
  If someone is able to receive data it must not be possible to replay this data later. The master and the slave must have the possibility to check that the data is not valid anymore.
- **R4: Fingerprinting must not be possible**
  If someone is able to receive all data he must not be able to correlate it with older data or to get information by using statistical tests.

Note: The communication itself can not be hidden, addresses (sender and receiver) are not confidential.

4 RFM12 Protocol

4.1 Media Access

Due to the fact that only one instance can send at a time one minute is divided into 60 time slots (00 to 59). For each slot only one sender is allowed to send his data.

All data transfer is initiated by the slave during his time slot. If the internal clock from a slave is not synced to the master it has to wait for at least one sync packet. This has two reasons: First reason is that the slave is only allowed to send during his dedicated time slot, second reason is that date and time is needed for security.

The slave sends a data packet to the master between 100 and 150ms of the time slot.

After that the master has the possibility to send one packet back to the slave during 400 and 450ms of the time slot.

The slave can reply to this packet by sending a response from 650 ms to the end of the time slot.

So a maximum of three packets can be send during one time slot.

Time slot for communication is dedicated by address.

- Without request from sync packet, slave can use only slot 01 to 29 (slot is same as address).
- With 4 byte communication request in sync packet, slave must use first next dedicated communication slot 01 to 29 or 31 to 29 (in slot same as address or address+30)
- With 1 byte communication request in sync packet, slave can use whole time for slot 31-59 independent to his address. Slave must start communication on time slot 31.

4.2 Synchronization Packets

At slot 00 and 30 the maser sends a sync frame.

The sync frame is used from the slaves to sync their internal clock to the global clock.

The sync packet contains date and time and additional data to request communication from the slaves.

The sync packet is shown in the following table:

<table>
<tr><th>Size</th><th>Content</th><th>Example</th></tr>
<tr><td>2+2 byte</td><td>Preamble (0xAAAA2DXX) where 0x2DXX is packet start pattern</td><td>0xAAAA2DD4</td></tr>
<tr><td>1 byte</td><td>Packet size
(bits 0 - 5 is packet size,
bit6=1 and bit7=1 to indicate sync packet)
Preamble and packet size byte is not calculated into this number.

example packet sizes:
With communication request:
- second 00: 4+4+4 = 12 byte
- second 30: 4+1+4 = 9 byte or
- second 30: 4+4+4 = 12 byte
Without communication request:
- second 00: 4+0+4 = 8 byte
- second 30: 4+0+4 = 8 byte
- second 00 or 30: 0+0+0 // dummy Sync packet, every 4th packet must be not dummy</td><td>0xCC
(Second 0 with request)

0xC8
(Second 30 without request)</td></tr>
<tr><td>4 byte</td><td>date and time

<table>
<tr><td>Byte 0</td><td>Y7</td><td>Y6</td><td>Y5</td><td>Y4</td><td>Y3</td><td>Y2</td><td>Y1</td><td>Y0</td></tr>
<tr><td>Byte 1</td><td>M3</td><td>M2</td><td>M1</td><td>M0</td><td>R2</td><td>R1</td><td>D4</td><td>D3</td></tr>
<tr><td>Byte 2</td><td>D2</td><td>D1</td><td>D0</td><td>H4</td><td>H3</td><td>H2</td><td>H1</td><td>H0</td></tr>
<tr><td>Byte 3</td><td>R0</td><td>Mi5</td><td>Mi4</td><td>Mi3</td><td>Mi2</td><td>Mi1</td><td>Mi0</td><td>S_{30}</td></tr>
</table>
Y - year from 2000 (0 - 255)
M - month (1 - 12)
D - day in month (1 - 31)
H - hour (0-23)
Mi - minute (0 - 59)
S_{30} - =0 → 0second ; =1 → 30second, sync packet is valid only in this seconds in minute
R - reserved, use 0</td><td>0x0911E92D
(2009/01/15 09:22:30)

0x0AC316E
(2010/12/24 22:55:00)</td></tr>
</table>

Size	Content	Example
0 to 4 byte	Request for communication - Sync packet time slot 00 or 30: can contain 4 byte flags for force communication in next time slot correspondent to address Byte 0 contain flags 07 to 00: // note: flag 00 is not used Byte 1 contain flags 15 to 08 Byte 2 contain flags 23 to 16 Byte 3: - bit 7 always 0 - bit 6 always 0 - bit 5 - bit 0 flag for communication 29-24 - Sync packet time slot 30 only: can contain 1 byte flag to block slot 31-59 for communication with selected slave - bit 7 always 1 - bit 6 always 1 - bit 5 - bit 0 selected slave address. This slave can communicate in any slot 31-59	0x2000000E (Sync 00: request 1,2,3,29) 0xCE (Sync 30: request slave 14)
4 byte	Message authentication Code (MAC) The MAC is calculated over the Data without preamble (Packet Size | date and time | Request for communication) according to chapter 5.3	

4.3 Data Packets

Slots 01 to 29 and 31 to 59 are used for communication from master to slave and vice versa.

All data is send in the following format:

Size	Content	Example
2+2 byte	Preamble (0xAAAA2DXX) where 0x2DXX is packet start pattern	0xAAAA2DD4
1 byte	Packet size (bits 0 - 5 is packet size, bit6=0 and bit7=0 to indicate data packet) Preamble and packet size byte is not calculated into this number. example packet sizes: - 4 byte data: 1+4+4 = 9 byte - 18 byte data: 1+18+4 = 23 byte	0x09 (9 byte data)

Size	Content	Example
1 byte	slave address and packet counter - bits 0 – 5: slave address - bits 6 - 7 packet counter in time slot during a time slot max 4 packets can be send: 0x03: slave 3 to master 0x43: master to slave 3 0x83: slave 3 to master	0x07 (slave 7 to Master)
2 - 18 byte	Data - encrypted The encryption is shown in chapter 5.4	
4 byte	Message authentication Code (MAC) The MAC is calculated over the Data without preamble (date and time | packet size | slave address | encrypted packet data) according to chapter 5.4	

5 Security

This chapter explains the security related measures. It shows which keys and algorithms are needed and which mode of encryption is used as well as how the message authentication code is computed. Furthermore it gives some hints how to distribute and store the keys.

5.1 Algorithm

Due to the small size of the flash in the HR20 a very small cipher must be chosen. Furthermore it would be useful to design the security in such a way, that only one direction of the cipher (only encryption, not decryption) has to be implemented in the HR20.

The cipher used for this project is XTEA [1] this is a very small block cipher with a block size of 64-bit and a 128-bit key.

We detonate the cryptographic operations in the following way:

cipherdata = encrypt_key (data): encrypt 64 bit data using the 128 Bit key

plaindata = decrypt_key (data): decrypt 64 bit data using the 128 Bit key

5.2 Keys

K_M = Master Key (128 Bit)

The whole system uses the same 128 bit key for one installation.

To reduce the overhead when setting individual keys for each installation an individual 128 bit key may be stored in the EEPROM File during compilation. Later this value can be altered using the service menu.

K_{MAC} = message authentication key (128 Bit)

K_{ENC} = encryption key (128 Bit)

Due to the fact that it isn't recommended to encrypt and generate message authentication codes with the same cryptographic key, we have to generate different keys out of the master key K_M. We use the following key derivation function to derivate the MAC key K_{MAC}

and the encryption key K_{ENC}

```
1. K_MAC = encrypt_K_M  (0xc7c6c5c4c3c2c1c0) |
     encrypt_K_M  (0xcfcecdcccbcac9c8)
2. K_ENC = encrypt_K_M  (0xcfcecdcccbcac9c8) |
     encrypt_K_M  (0xd7d6d5d4d3d2d1d0)
```

The keys K_{MAC} and K_{ENC} can be either calculated at boot time and every time when K_M was altered. It is also possible to store K_{MAC} and K_{ENC} in the EEPROM of the HR20.

5.3 MAC

This section shows how the message authentication code is computed.

In a first step it must be ensured that each data is unique so that it is not possible to resend the same packet later. This is different for both packet types (sync and data).

A sync packet contains the actual date and time so each sync packet is unique, no additional data must be added.

MAC = MAC_K_{MAC} (packet size | date and time | request for communication)

Data packets are send in a dedicated time slot identified by date and time. Each packet inside a time slot is unique due to the packet counter bits in the slave address. To calculate the message authentication code date and time are concatenated with the packet data.

MAC = MAC_K_{MAC} (date and time | packet size | slave address | encrypted packet data)

A MAC is computed according the CMAC Algorithm [CMAC] with the underlaying block cipher XTEA.

For MAC wee need two subkeys K_1 and K_2 which are derived from K_{MAC} using the following algorithm.

```
1. L = encrypt_K_MAC  (0x0000000000000000)
2. If MSB (L) = 0, then K1 = L << 1;
     else K1 = (L << 1) XOR 0x000000000000001B
3. If MSB(K1) = 0, then K2 = K1 << 1;
       else K2 = (K1 << 1) XOR 0x000000000000001B
```

The subkeys K1 and K2 can be either calculated at boot time and every time when K_M was altered. It is also possible to store K1 and K2 in the EEPROM of the HR20. Note: K1 and K2 have only 64 bit length each.

The MAC (DATA) is calculated in the following way, according to CMAC with cipher block length of 64 bit:

```
1. Let Mlen = message length in bits
2. Let n = Mlen / 64
3. Let M1, M2, ... , Mn-1, Mn
   denote the unique sequence of bit strings such that
   M = M1 || M2 || ... || Mn-1 || Mn,
   where M1, M2,..., Mn-1 are complete 8 byte blocks.
4. If Mn is a complete block, let Mn = K1 XOR Mn else,
   let Mn = K2 XOR (Mn ||10j), where j = n*64 - Mlen - 1.
5. Let C0 = 0
```

```
6. For i = 1 to n, let Ci = ENC_KMAC(Ci-1 XOR Mi).
7. Let MAC = MSB32(Cn). (4 most significant byte)
8. Return MAC
```

The MAC is verified in the same way, The receiver has to compute the same data with the same keys and after that he verifies that the received MAC is equal to the computed MAC. If not, than the message is not authentic. This implicates the functionality of a error correction code like CRC which is not needed anymore.

Note: The MAC is computed over the packet as it is send (including date and time) especially when the data is encrypted the MAC is computed over the encrypted data. Decryption is applied to the data after the MAC was checked.

5.4 Encryption

As the functions implemented in the HR20 shall be reduced, we set up the encryption in such a way, that only XTEA encryption is needed in the HR20. Therefore we use the CTR mode of operation with Nonce = (date and time – 6 byte) | (block counter – 2 byte).

Block counter resets for each communication window.
Date and time is from start of communication window.

As maximum 18 Byte of data have to be encrypted the cipher has to be involved maximum three times in the following way:

```
1. Let nonce = date and time - 6 byte
2. Let S64 = ENC_KENC_(nonce|counter++)
3. Let CIPHERTEXT64 = PLAIN64 XOR S64
4. repeat from 2. for each 64 bits (8 bytes)
5. Return CIPHERTEXT
```

The decryption is done in the same way, so we don't need the XTEA decryption function.

[CMAC] http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

[CTR] http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[XTEA] http://en.wikipedia.org/wiki/XTEA