Insteon' False Security And Deceptive Documentation

Peter Shipley
Ryan Gooler

Outline

- Intro
- What is Insteon
- Insteon’s Security Model
- Insteon’s documented vs. actual protocol
- How I did it
- Live Demo
- Q & A
- Drinks (in bar after talk)

I will focus on Insteon’s security model and wireless protocol as well as the inaccuracies in Insteon’s well published documentation.

I did not examine the power line communications as I had no desire to connect analyzing gear to 120V

What Is Insteon

“Insteon is a home automation technology that enables light switches, lights, thermostats, motion sensors, and other electrical devices to interoperate through power lines and/or radio frequency (RF) communications.*

* Wikipedia

Insteon uses a “Dual-band communications using both the powerline and the airwaves”

Almost all Insteon device are capable of operating as a repeater / bridge.

Note: Older and battery operated devices do not repeat.

What Is Insteon

Insteon is supported by most home automation controllers in addition to :

Google’s Nest

Microsoft’s Windows 8.1 “Live Tile”

Apps Android, IOS & Apple Watch

What Is It Good For?

Communicating and Controlling

- Lights
- Sprinklers
- Locks
- Alarms Systems
- I/O Controllers
- Remote sensors
- Thermostats
- Home monitoring

Weakest Link?

While Insteon is primarily a automation protocol used for lighting and environment, it is being integrated into more advanced systems potentially becoming the weakest link.

Published Specifications

Insteon is nice enough to publish several papers detailing their product, one is entitled :

"WHITEPAPER: The Details"

This document covers the products' layer 1 and 2 protocols and security in high detail.

Google: "Insteon whitepaper" for your very own copy

There is also a publication called

“WHITEPAPER: Insteon compared”

which is a thinly veiled marketing document comparing Insteon’s product to ZigBee and Zwave.

After reading the specs if occurred to me that it would easily be able to write a unix based insteon client with common tools like rfcat

All attempts failed…. and I wondered why….

This piqued my interest….

So…. I investigated….

After successfully reverse engineering the Insteon RF protocol I discovered major parts of their “Whitepaper" were **almost a complete work of fiction**.

The Insteon published protocol documentation had so many deceptive entries I ran out of synonyms for the word “Bullshit” to use in these slides

Fortunately I have a lot of multilingual friends and co-workers

Published RF Specification
(Layer 2)

Center Frequency	915Mhz	**Bullshit**
Encoding Method	Manchester	**Bullshit**
Modulation	FSK	**TRUE!!**
Deviation	64,000 Hz	**Bullshit**
Symbol Rate	76,800 sym/s	**Bullshit**
Data Rate	38,400 bits/s	**Bullshit**

*** 83% BULLSHIT ***

* WHITEPAPER: The Details

The Insteon Documentation describes FSK as follows :

> “Symbols are modulated onto the carrier using frequency-shift keying (FSK), where a zero-symbol modulates the carrier half the FSK deviation frequency downward and a one-symbol modulates the carrier half the FSK deviation frequency upward*

*** [illegible] ***

* WHITEPAPER: The Details

To be accurate :

FSK deviation is equal to the absolute value of the difference between the center frequency and the mark or space frequencies.

What they were describing is “Shift”

“deviation” is equal, to one-half of the shift

(To be honest, this probably actually was a typo)

* WHITEPAPER: The Details

Actual RF Specification
(Layer 2)

Center Frequency	~~915Mhz~~	914.975 Mhz
Encoding Method	~~Manchester~~	“Tokenized” Manchester
Modulation	FSK	FSK (inverted)
~~Deviation~~ (Shfit)	~~64,000 Hz~~	150,000 Hz
Symbol Rate	~~76,800 sym/s~~	9125 sym/s
Data Rate	~~38,400 bits/s~~	2600 bit/s

Standard Packet format

“RF messages begin with two sync bytes consisting of AAAA in hexadecimal, followed by a start code byte of C3 in hexadecimal

AA	AA	C3	X	X	X	X	X	X	X	X	X	X
Preamble		Sync	From Addr			To Addr			Flag	Cmd	opt	CRC

*** HOGWASH ***

* WHITEPAPER: The Details

Extended Packet format

AA	AA	C3	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Pre		Sync	From		To			Flag	DATA																		CRC	

*** Malarkey ***

* WHITEPAPER: The Details

Actual Packet Order

X	X	X	X	X	X	X	X	X	X
Flag	To Addr			From Addr			cmd	opt	crc

Each byte (X) is encoded as 26 bits:

- “11” followed by
  - + 5 bit index number (manchester encoded, LSB)
  - + 8 bit byte (manchester encoded, LSB)

Packets can be optionally padded to the full 13 or 32 bytes

Byte	Index + Dat	(LSB)	Manchester Encoded
0x03⇒	11111 00000011⇒	11111 11000000⇒	0101010101 0101101010101010
0xE5⇒	01011 11100101⇒	11010 10100111⇒	0101100110 0110011010010101
0x3F⇒	01010 00111111⇒	01010 11111100⇒	1001100110 0101010101011010

Then preamble is prepended (0xAA) and the data is **INVERTED**

```
1010101000101010101010100101010101010100101010101010100101010101
0010100110011001100101101010
```

(Note this represents only the first 3 bytes of a packet)

Packet Flags

Bit	Flag	Meaning
Bit 7	Message Type	100 = Broadcast Message 000 = Direct Message 001 = ACK of Direct Message 101 = NAK of Direct Message 110 = Group Broadcast Message 010 = Group Cleanup Direct Message 011 = ACK of Group Cleanup Direct Message 111 = NAK of Group Cleanup Direct Message
Bit 6		
Bit 5		
Bit 4	Extended	1 = Extended Message 0 = Standard Message
Bit 3	Max Left	00 = 0 message retransmissions remaining 01 = 1 message retransmission remaining 10 = 2 message retransmissions remaining 11 = 3 message retransmissions remaining
Bit 2		
Bit 1	Max Hops	00 = Do not retransmit this message 01 = Retransmit this message 1 time maximum 10 = Retransmit this message 2 times maximum 11 = Retransmit this message 3 times maximum
Bit 0		

* WHITEPAPER: The Details

Message Integrity Byte

"INSTEON uses a software-implemented 7-bit linear-feedback shift register with taps at the two most-significant bits*

Unsurprisingly, after applying a myriad of tool for shift-register analysis it was clear that this was not the case

*** Poppycock ***

* WHITEPAPER: The Details

what Insteon implemented is not even a true shift registar

For each byte:

- XOR with the running checksum
- XOR the upper nibble with the lower nibble
- Shift lower nibble by one then XOR with upper nibble

```
r = 0 ;
for(i=0;i<dat_len;i++) {
  r ^= dat[i] ;
  r ^= (( r ^ ( r << 1 )) & 0x0F) << 4 ;
}
```

(Sometime methods are simpler to describe in code)

Security

“INSTEON network security is maintained at two levels. Linking Control ensures that users cannot create links that would allow them to control their neighbors’ INSTEON devices*

Linking Control

“INSTEON enforces Linking Control by requiring that users have Physical Possession of Devices in order to create links *

*** ФИГНЯ ***

* WHITEPAPER: The Details

Security

Physical Possession of Devices

> “Firmware in INSTEON devices prohibits them from identifying themselves to other devices unless a user physically presses a button on the device*

Masking Non-linked Network Traffic

> “firmware masks the two high-bytes in the address fields of INSTEON messages unless the traffic is from an INSTEON device already linked*

*** בולשיט ***

* WHITEPAPER: The Details

Security?

The Insteon security model relies on security through obscurity, primarily that unique 3 byte device addresses can not be intercepted with their firmware.

(Reminder: Insteon devices communicate via unencrypted protocols)

Encryption

INSTEON documentation contains multiple references encryption support.

INSTEON Encryption within Extended Messages

"For applications such as door locks and security systems, INSTEON Extended messages can contain encrypted payloads. Possible encryption methods include rolling-code, managed-key, and public-key algorithms*

*** rahat ***

* WHITEPAPER: The Details

"INSTEON extended messages allow for encryption using global standards such as AES-256, the same as with other protocols*

*** skitsnack ***

* WHITEPAPER: Insteon compared

While the documentation never outright says it encrypts the documentation strongly implies it

INSTEON itself **does not directly support encryption**, if is does not have not seen it nor have any direct references in the protocol.

Reverse Engineering

Reverse Engineering The Checksum

The Checksum is documented as an :

> “software-implemented 7-bit linear-feedback shift register"

As this was not the case i could not use standard CRC tools to analyze the data and had to do it by hand

(luckily we are dealing with 8bit data)

Basic analysis indicated the checksum was a permutation of rolling xor :

```
03 64 78 24 80 25 13 19 00 14 = 80
07 E5 3F 16 80 25 13 19 00 B4 = D0
0B 69 54 17 80 25 13 19 00 CE = 40
```

Notice the final value always ends in zero

This Leaves only the high nibble to be derivatived.

Next I determined is their algorithm was stateless or not

(that is the CRC engine only acts upon the current byte (or word) without regard of surrounding data)

```
00 00 00 00 00 00 00 00 10 : 10
00 00 00 00 00 00 00 20 00 : 20
00 30 00 00 00 00 00 00 00 : 30
00 00 40 00 00 00 00 00 00 : 40
00 00 00 50 00 00 00 00 00 : 50
```

I was able to verify this by varying one bit in a packet and observing the difference in the xor’ed sum.

Next I varied low order bits:

```
.. .. .. .. .. .. .. 01 .. 31
.. .. .. .. .. .. .. 02 .. 62
.. .. .. .. .. .. .. 03 .. 53
.. .. .. .. .. .. .. 04 .. C4
.. .. .. .. .. .. .. 06 .. A6
.. .. .. .. .. .. .. 07 .. 97
.. .. .. .. .. .. .. 08 .. 88
.. .. .. .. .. .. .. 09 .. B9
.. .. .. .. .. .. .. 0A .. EA
.. .. .. .. .. .. .. 0B .. DB
.. .. .. .. .. .. .. 0C .. 4C
.. .. .. .. .. .. .. 0D .. 7D
.. .. .. .. .. .. .. 0F .. 1F
```

(in the above data “00” was replaced with “..” for easier viewing)

This clearly showed a one to one mapping between lower and upper nibbles:

```
0000 0000 -> 0000 0000
0000 0001 -> 0011 0000
0000 0010 -> 0110 0000
0000 0011 -> 0101 0000
0000 0100 -> 1100 0000
0000 0101 -> 1111 0000
0000 0110 -> 1010 0000
0000 0111 -> 1001 0000
0000 1000 -> 1000 0000
```

Translated to in C:

```
r = 0 ;
for(i=0;i<dat_len;i++) {
  r ^= dat[i] ;
  r ^= (( r ^ ( r << 1 )) & 0x0F) << 4 ;
}
```

Reverse Engineering The Protocol

Aside from using standard procedure for identifying the RF signals, There are no real step by step instructions or magical tools for cracking actual protocol

Sorry

Tools

I have written a hand full of tools designed to be modular and reusable (I hope).

Tools

Hardware :

- rtl_sdr (receive only)
- rfcat
- hack-rf (patched for stdin/stdout)

To Receive :

```
rtl_reciv.sh | fsk2_demod | print_pkt.py
```

or

```
hackrf_reciv.sh | fsk2_demod | print_pkt.py
```

or

```
rf_reciv.sh | print_pkt.py
```

To Transmit :

```
send_comm.py -r 07 E5 3F 16 80 25 13 11 BF 13
00 00 AA | fsk2_mod | hackrf_xmit.sh
```

or

```
send_comm.py -r 07 E5 3F 16 80 25 13 11 BF 13
00 00 AA | rf_xmit.sh
```

Realtime Demonstration

Copyrights

INSTEON is a trademark of INSTEON.

“WHITEPAPER: The Details” :

Thank You

- David Woolsey : Friend and Physicist
- Zac Franklin : Encouragement
- Major Malfunction : Radio Questions
- Andy Beals : Old friend with a PowerPC
- Mike Walters : https://github.com/miek/