Reverse Engineering the PowerG Wireless Protocol

James Chambers and Sultan Qasim Khan

REcon 2024 – June 29, 2024

What is PowerG?

- Proprietary radio protocol for wireless security and safety systems
  - Developed by Johnson Controls (formerly Tyco / DSC)
- **Smart locks**: remote locking/unlocking, reporting status
- **Sensors**: report their status to alarm system panels
  - Door and window contact sensors, motion detectors, window break sensors, smoke detectors
- **Alarm system panels**: trigger sirens and lights
- **Key fobs**: arm/disarm alarm systems or trigger the alarm

PowerG Specs

- Employs frequency hopping, adaptive transmission power, and encryption for system security and reliability
- North American PowerG devices operate in the 915 MHz ISM band
- Range:

PowerG Security Claims - Encryption

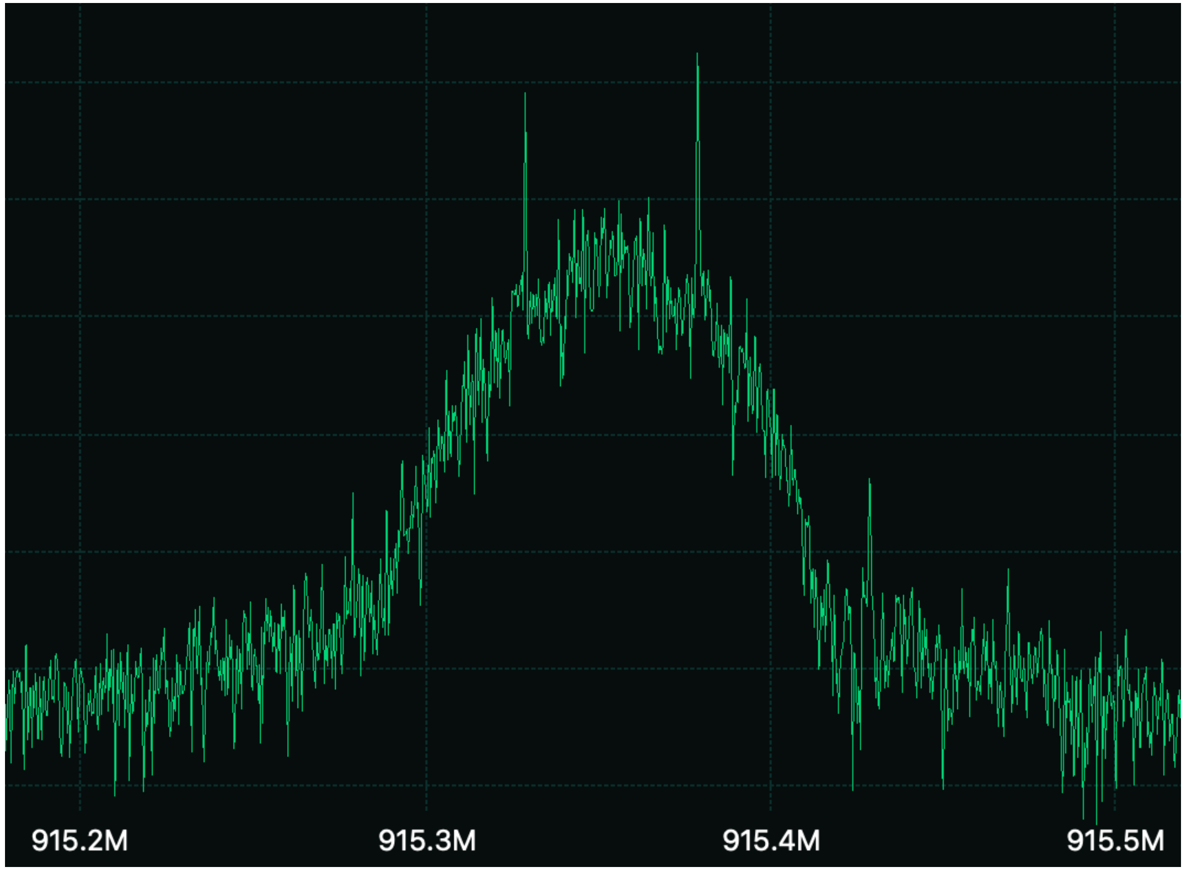
- "PowerG uses 128-bit AES advanced encryption to protect you from devious intruders, code grabbing, and message substitution from hackers."
  - Source: "PowerG - The power of wires, without the wires"
  - https://www.youtube.com/watch?v=8hzX90NQWcg
- "AES is a well-proven encryption algorithm that guarantees strong authentication and encryption security for the PowerG wireless network."
  - Source: PowerG Technology Overview
  - https://cms.dsc.com/download.php?t=1&id=24255

PowerG Security Claims – Frequency Hopping

- "FHSS changes the frequency of a transmission at intervals faster than an intruder can retune a jamming device."
- "Once a wireless connection is established and time-synchronization is gained, the receiver and transmitter agree on one of practically infinite frequency hopping sequences. These sequences are both encrypted and time-dependent."
- "Unless the system time, the system encryption key and the proper calculation are all known, the communication cannot be tracked. As a result, unauthorized interception of, or eavesdropping on, a communication is virtually impossible."
- Source: PowerG Technology Overview
  - https://cms.dsc.com/download.php?t=1&id=24255

Initial Black Box Analysis – Radio Modulation

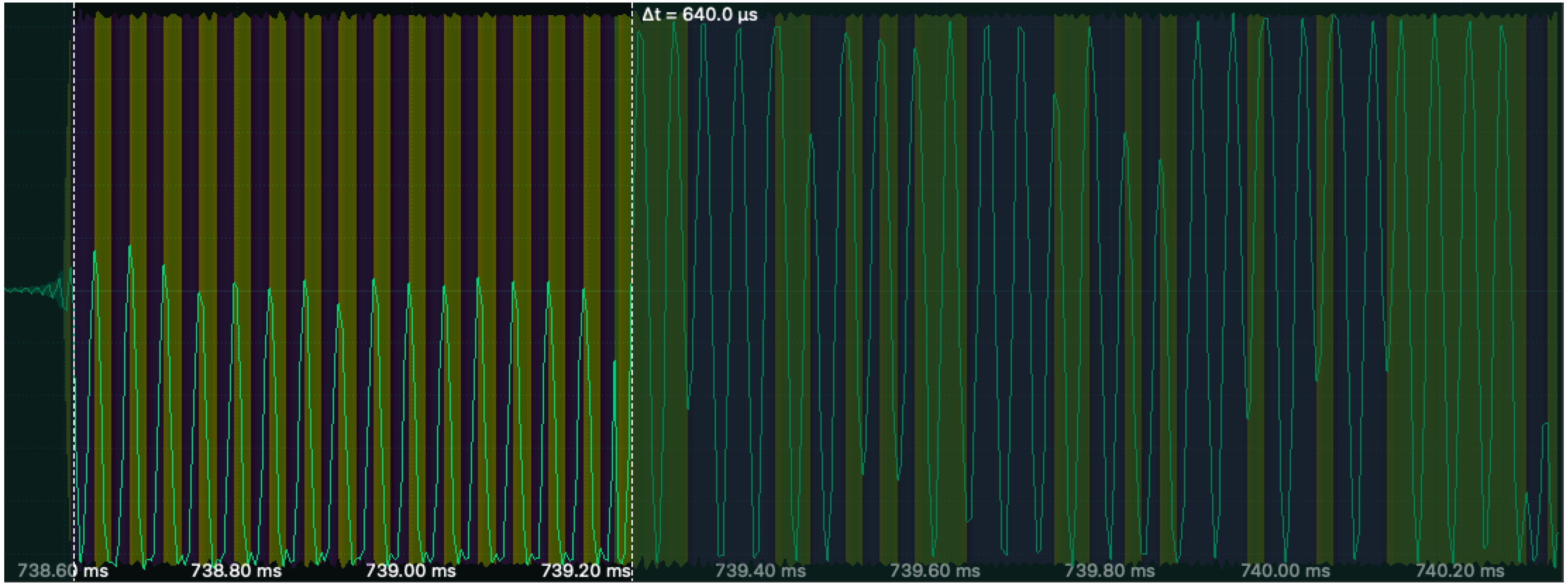
- Gaussian Frequency Shift Keying (GFSK) modulation
  - Can be seen in rounded shape of modulation in power spectral density (PSD) plot
- 25 kHz deviation
  - Peaks in power spectrum are 50 kHz apart



PSD of a Single PowerG Radio Packet

Initial Black Box Analysis – Symbol Rate

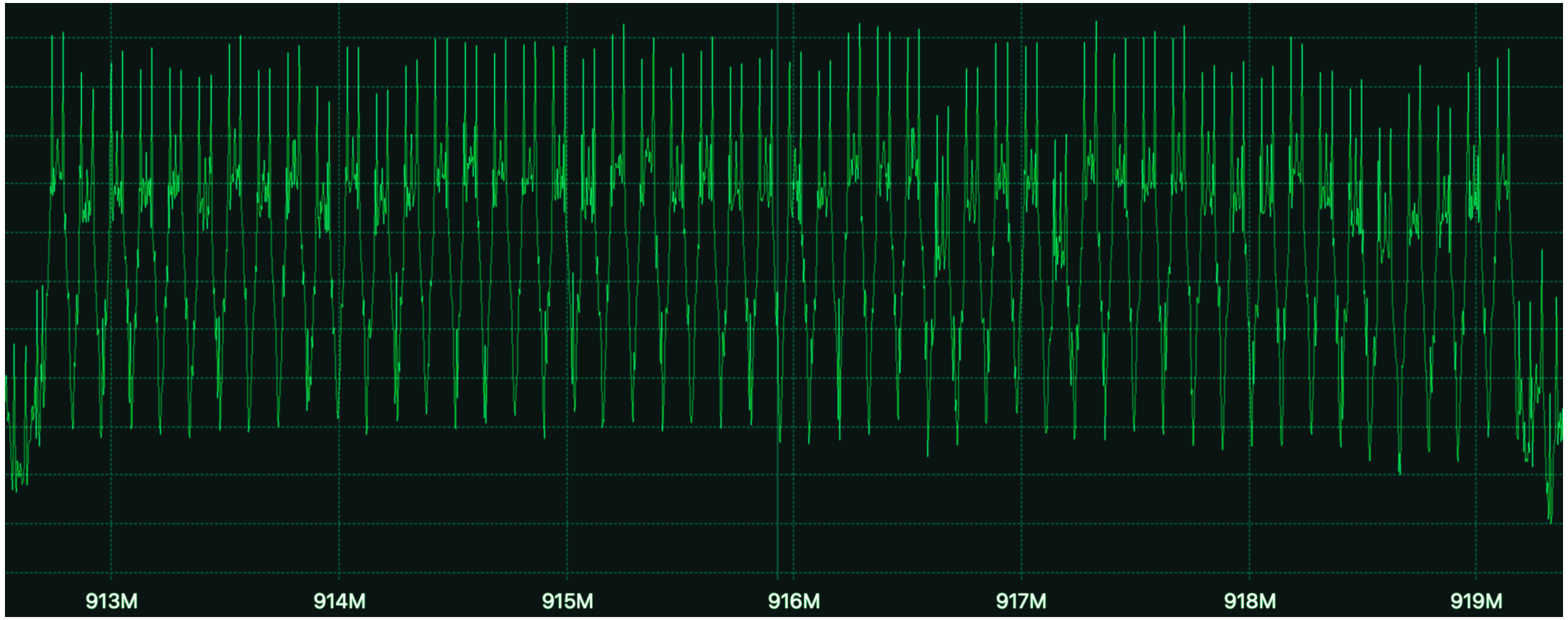
- 50000 symbols per second
  - 640 us for a 32-bit preamble of alternating 1s and 0s



In-phase component of a time domain preview of the baseband signal at the start of a PowerG radio packet. Background colour highlighting shows the frequency (derivative of phase).

Initial Black Box Analysis – Frequency Hopping

- 50 channels observed from 912.750 to 919.106 MHz
  - Spaced 129.73 kHz apart
- Random-looking hop sequence with 64 hops per second



PowerG Peak Power Spectral Density Plot

Initial Black-Box Analysis – Packet Structure

- We used Universal Radio Hacker to analyze captured traffic for patterns
- Observed consistent start to every packet:
  - 32-bit preamble of 101010..1010 (0xAAAAAA in MSB-first format)
  - 32-bit sync word of 0x1F351F35 (in big-endian MSB-first format)
- Various values repeated for the next 16 bits, suggesting a header but interpretation was non-obvious at first
  - Upon applying TI CC1101-style dewhitening (supported by URH), it became clear that the first byte after the sync word was a length field
  - Next two bytes after length field had values that often repeated, suggesting they may be addresses

Correspondence with TI Standard Packet Format

- The observed packet structure matched the TI standard packet format with CC1101-style whitening, 1 byte length field, and 1 byte address
- CC1101-style CRC calculations over the header and payload matched the last 16 bits of every packet

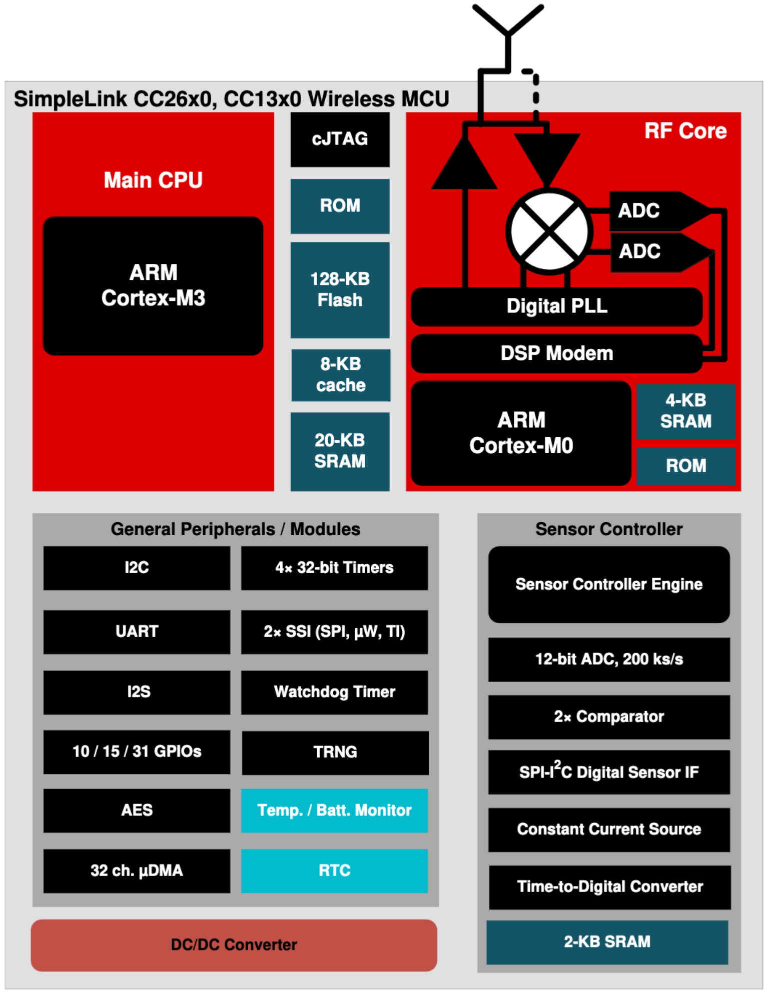
For compatibility with existing TI parts, the packet format given in Figure 25-9 can be used in most cases. This packet format is supported through the use of the commands CMD_PROP_TX and CMD_PROP_RX.

1 bit to 32 bytes	8 to 32 bits	0 or 1 byte	0 or 1 byte	0 to 255 bytes	0 or 16 bits (0 to 32 bits)
Preamble	Sync word	Length field	Address	Payload	CRC

Figure 25-9. Standard Packet Format

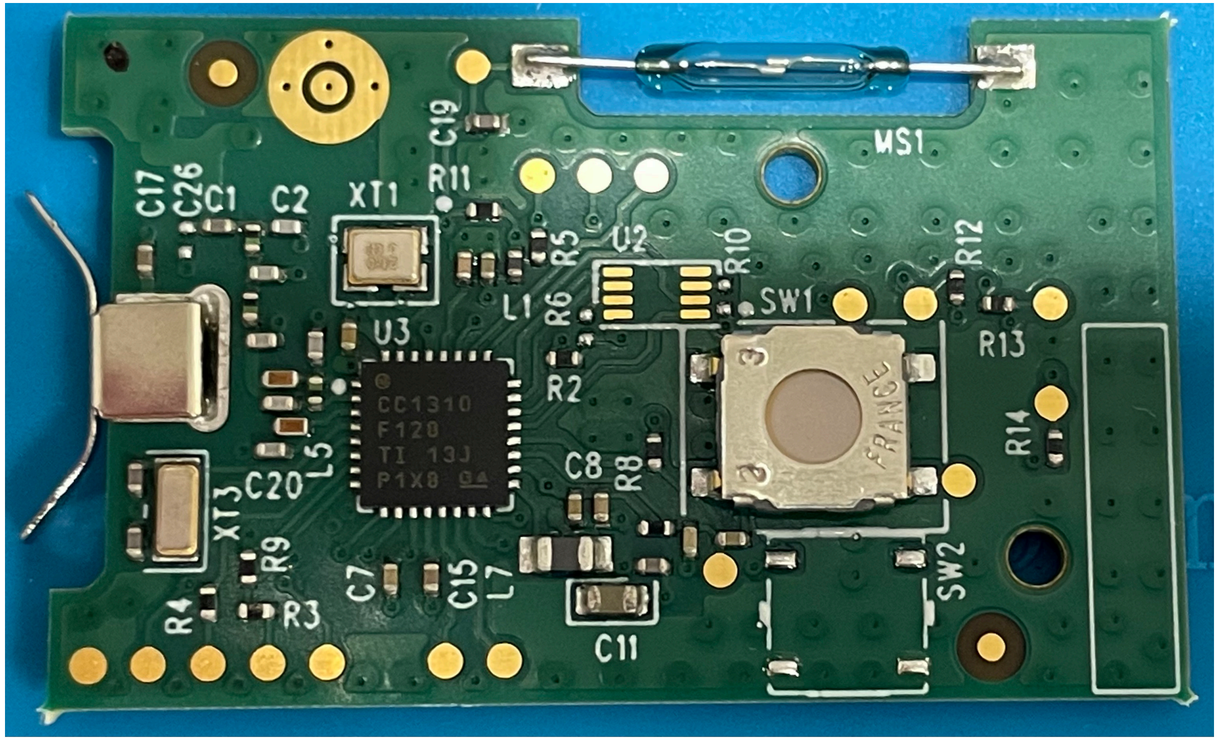
PowerG Modem Hardware and Firmware

- PowerG modems run on Texas Instruments sub-1 GHz wireless MCUs, incl. CC1110 and CC1310
- Recent models use the CC1310
  - ARM Cortex-M3 main CPU running from flash memory
  - ARM Cortex-M0 radio CPU running from ROM
  - Supports implementing custom/proprietary radio protocols with a set of flexible radio commands
- PowerG modem firmware found to be built with TI SimpleLink SDK, and running on top of TI-RTOS



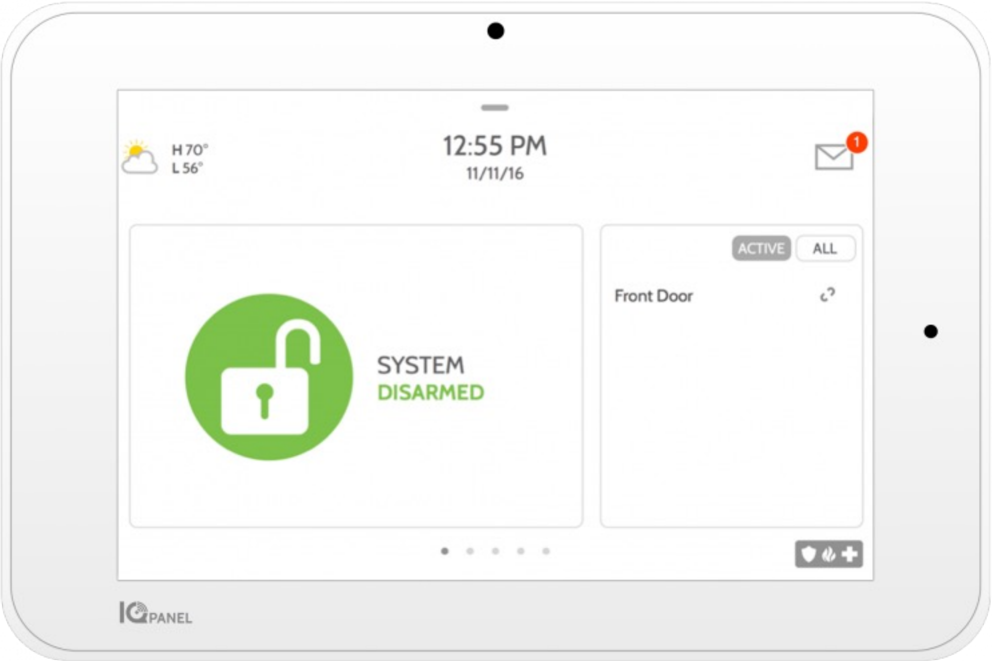
PowerG Modem Hardware and Firmware

- Some sensor devices run entirely on the TI microcontroller

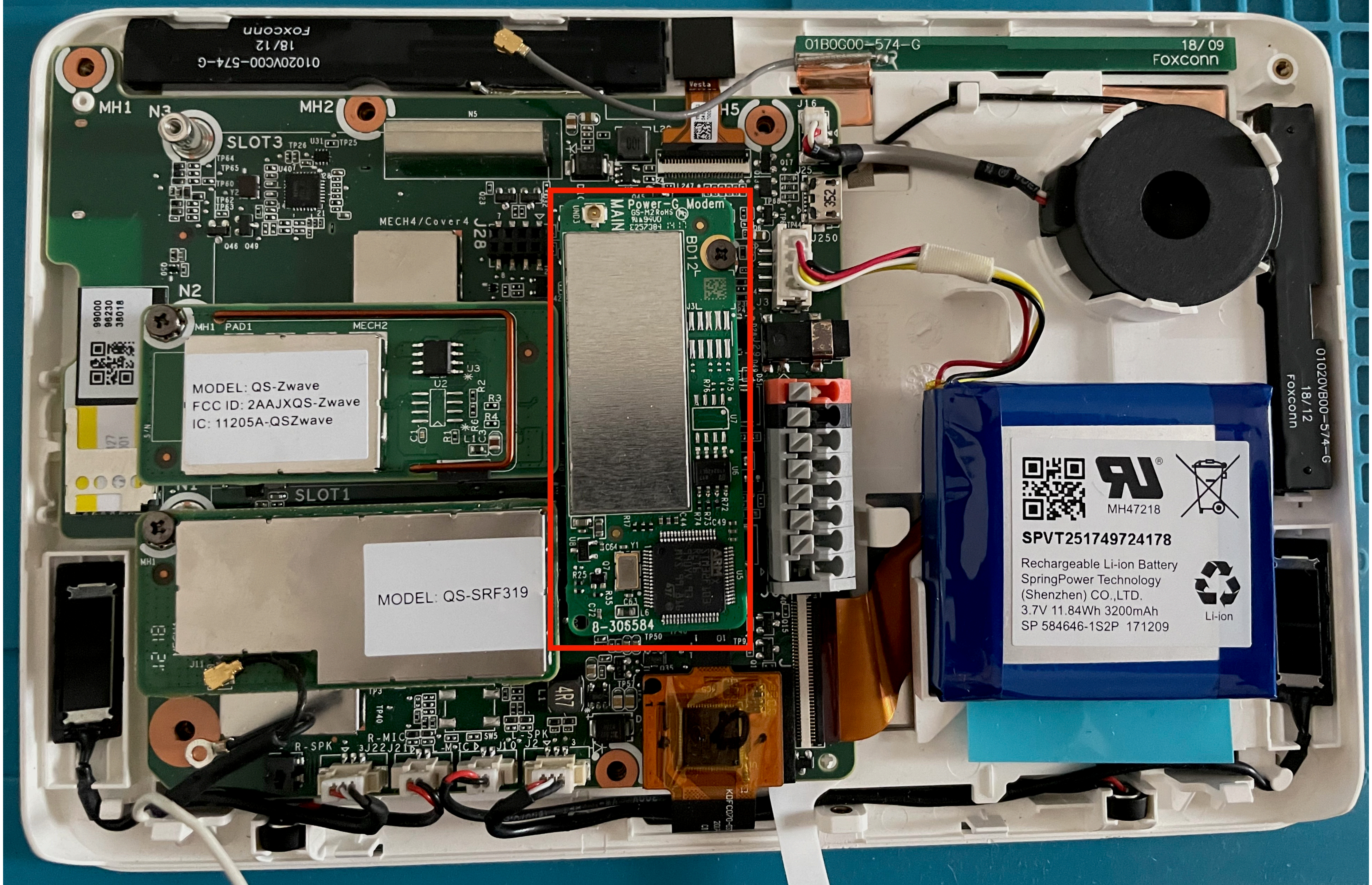


PowerG Modem Hardware and Firmware

- Android-based control panels use PowerG daughterboard as a modem
- Daughterboard modem talks to "virtual modem" service via UART

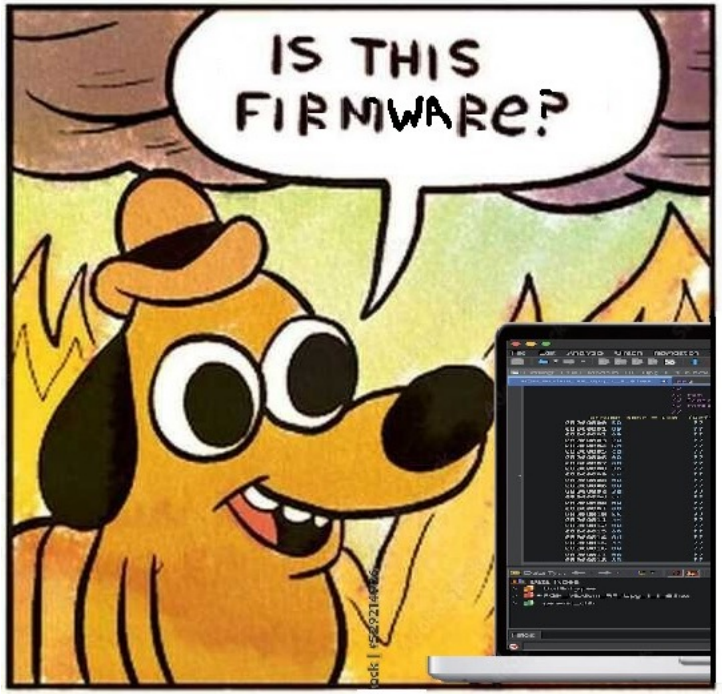


nccgroup


Foxconn
18/12
01020VC00-574-G
01B0G00-574-G
18/09
Foxconn
MH1
N3
MH2
SLOT3
H5
J16
Power-G Modem
MAIN
GS-M2 RoHS
BD12
MECH4/Cover4
J28
N2
MH1
PAD1
MECH2
MODEL: QS-Zwave
FCC ID: 2AAJXQS-Zwave
IC: 11205A-QSZwave
SLOT1
MH1
MODEL: QS-SRF319
8-306584
MH47218
SPVT251749724178
Rechargeable Li-ion Battery
SpringPower Technology
(Shenzhen) CO.,LTD.
3.7V 11.84Wh 3200mAh
SP 584646-1S2P 171209
Li-ion
01020VB00-574-G
18/12
Foxconn
R-SPK
R-MIC
L-SPK
4R7

Initial Firmware Analysis

- Started analyzing PowerG as a research project in early 2023, without hardware
- Aware of PowerG related firmware included in OTA updates for control panel
- Multiple firmware images, apparently for different devices and different chips on one device
- All bare metal firmware, with no symbols & very few strings
  - No application-related strings in the actual target firmware
- Multiple versions of PowerG daughterboards registered with FCC
  - Newer ones keep details confidential
  - Can't make out chip IDs on internal photos



Initial Firmware Analysis

- All we know is they all look like ARM, and one of them is for an STM32
- Used Symgrate to try to figure out what each image is for
  - Fingerprint chip with peripheral accesses, recover function symbol names
- Symgrate finds a handful of function names, including:

  `NOROM_ThisLibraryIsFor_CC13x0_HwRev20AndLater_HaltIfViolated`

- Identifies that this firmware is for the CC13x0 chip and uses TI RTOS / TI SDKs
- Knowing the chip, load SVD definitions to define MMIO peripherals
  - Identifies like MMIO RF & cryptography commands
- Fill in more standard TI RTOS library functions, recover structure of tasks and mailboxes…

IAR Data Segment Compression

- The data segment (initial values for variables) referred to by the region table appeared to be stored in a compressed format
- Early boot code was found to decompress this segment when loading it from flash to RAM
- Review of IAR documentation and analysis of the decompression algorithm suggested its an IAR encoding of the LZ77 algorithm
- We reimplemented the decompressor in Python to reconstruct the initial values for variables in the data segment

Radio Operation Logic in Firmware

- TI CC13x0 hardware provides set of radio commands defined in ROM
- Radio command parameter structures passed to radio coprocessor (Cortex-M0) over mailbox interface
- Parameter structs located by searching data section for sync word 0x1F351F35
- Radio control logic identified through references to parameter structs

2000053c	02 38	uint16_t	3802h	commandNo
2000053e	00 00	zzz_radi…	IDLE	status
20000540	00 00 00 00	RF_Op *	00000000	pNextOp
20000544	00 00 00 00	uint32_t	0h	startTime
20000548	80	uint8_t	80h	startTrigger
20000549	01	uint8_t	01h	condition
2000054a	3c	rfc_CMD_…		pktConf
2000054b	eb	rxConf_t		rxConf
2000054c	35 1f 35 1f	uint32_t	1F351F35h	syncWord
20000550	ff	uint8_t	FFh	maxPktLen
20000551	01	uint8_t	01h	address0
20000552	ff	uint8_t	FFh	address1
20000553	04	uint8_t	04h	endTrigger
20000554	ec a9 00 00	uint32_t	A9ECh	endTime
20000558	00 00 00 00	dataQueu…	00000000	pQueue
2000055c	00 00 00 00	void *	00000000	pOutput

Cryptography Operations in Firmware

- AES ECB encrypt is the only hardware-based cryptography op used in the modem firmware
- Used for software AES-CTR
- CTR mode only needs the basic block encryption op
  - Encrypts blocks containing a counter and nonce value to generate keystream
  - Keystream used for XOR encryption & decryption

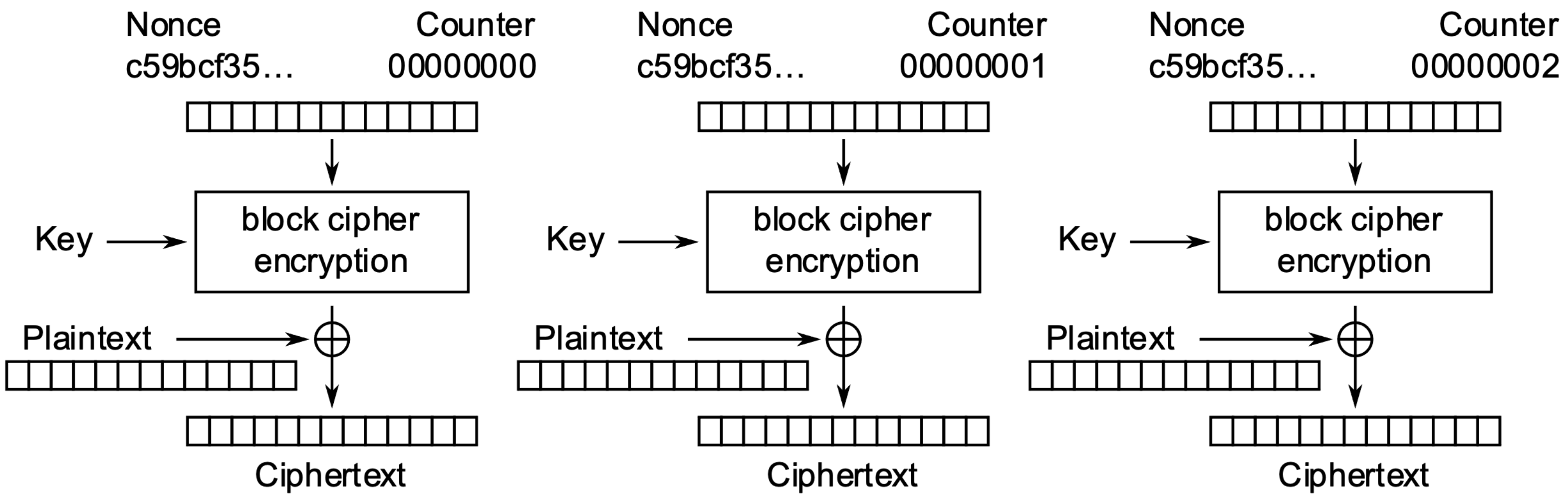
Decompile: CRYPTOAesEcb - (QOLSYS_V3_mode...

```
                    /* KEY_SIZE = 0b01 (128 bit), DIR = 0b1 (encrypt) */
    if (bEncrypt) {
      aes_ctl = 0b00001100;
    }
    else {
                    /* KEY_SIZE = 0b01 (128 bit), DIR = 0b0 (decrypt) */
      aes_ctl = 0b00001000;
    }
    Peripherals::CRYPTO.AESCTL = aes_ctl;
    Peripherals::CRYPTO.AESDATALEN0 = 0x10;
    Peripherals::CRYPTO.AESDATALEN1 = 0;
    _DAT_42480000 = 1;
    Peripherals::CRYPTO.DMACH0EXTADDR = (uint)pui32MsgIn;
    Peripherals::CRYPTO.DMACH0LEN = 0x10;
    _DAT_42480400 = 1;
    Peripherals::CRYPTO.DMACH1EXTADDR = (uint)pui32MsgOut;
    Peripherals::CRYPTO.DMACH1LEN = 0x10;
    uVar2 = 0;
  }
  return uVar2;
```


nccgroup

PowerG AES-CTR

- Normally, AES-CTR operates by encrypting blocks containing a nonce combined with an incrementing counter value

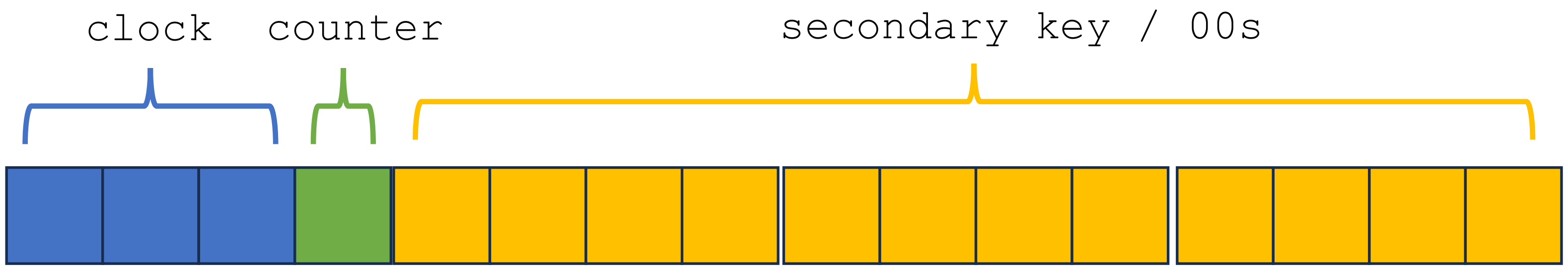


Counter (CTR) mode encryption

https://en.wikipedia.org/wiki/File:CTR_encryption_2.svg

PowerG AES-CTR

- PowerG uses a slight variation of AES-CTR, where the encrypted 128-bit block contains:
  - 24-bits containing the 32 kHz modem clock rounded to 1/64th of a second (only 23 bits are significant)
  - 8-bit incrementing counter
  - 96-bit fixed “secondary key”, or 96 zero bits



nccgroup

PowerG AES-CTR

- This construct is reused for several purposes:
  - Packet encryption
  - Validating that clocks used for encryption match
    - (directly output keystream bytes)
  - Channel hopping
    - (also based on directly outputting keystream bytes)

Packet Encryption

- Generate keystream check value by using initial counter value 0xFF
- Encrypt packet data with initial counter starting at 0x00
  - Increments as necessary, but the RF packets are generally small
- RF packets have four cryptography modes:
  - 0: not encrypted
  - 1: not encrypted
  - 2: encrypted using the 96-bit all-zero "nonce"
  - 3: encrypted using the 96-bit secondary key "nonce"

Channel Hopping

- Generate AES-CTR keystream output with initial counter = 0xFE
- Extract fifth and sixth bytes of keystream: call them A and B
- Calculate channel ID based on hop config:
  - 0: A % 50
  - 1: (A + 25) % 50
  - 2: B % 50
- Hop config 2 only updates every 4 seconds
  - (mask AES-CTR clock with ~0x1ffff)
- There is also a static channel, 15 (zero-based count)
  - Can still transmit when device clocks are too far out of sync

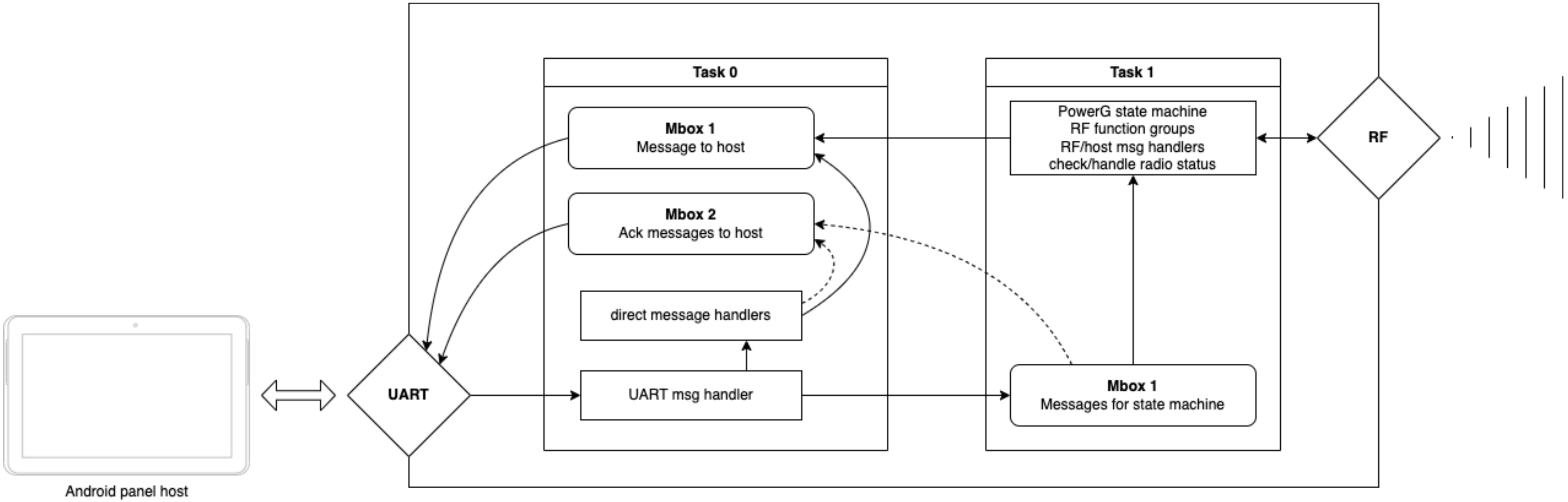
PowerG AES-CTR Problems

- It's important that the nonce is unique in CTR mode
  - Those aren't nonces
  - 23-bit 64 Hz clock can only count for ~36 hours until it wraps around to 0
  - Each time a clock value repeats, the same keystream is generated
- Effectively repeated-key XOR, can be broken statistically to recover plaintext (though these plaintexts are relatively predictable)
    - Cryptopals set 3, challenge 20
- Encrypted packets you capture passively can be used for active attacks against the encryption on that network in the future, each time the corresponding clock value repeats

PowerG AES-CTR Problems

- Captured messages can be replayed when clock repeats
- *Modified* captured message can be replayed when clock repeats
  - No cryptographic authentication
  - CTR mode is malleable since it's based on XOR encryption
    - (Remember, AES is just used to generate blocks of keystream)
  - Ciphertext of messages with known structure can be directly manipulated
    - Flipping one bit in the ciphertext causes corresponding bit flip in plaintext
  - Known plaintext attack: ciphertext XOR plaintext = keystream
    - Sensors are sending well-structured data that is mostly the same across devices of same model
    - Recover all/most of keystream to encrypt arbitrary messages on clock repeat

Overall Modem Firmware Architecture

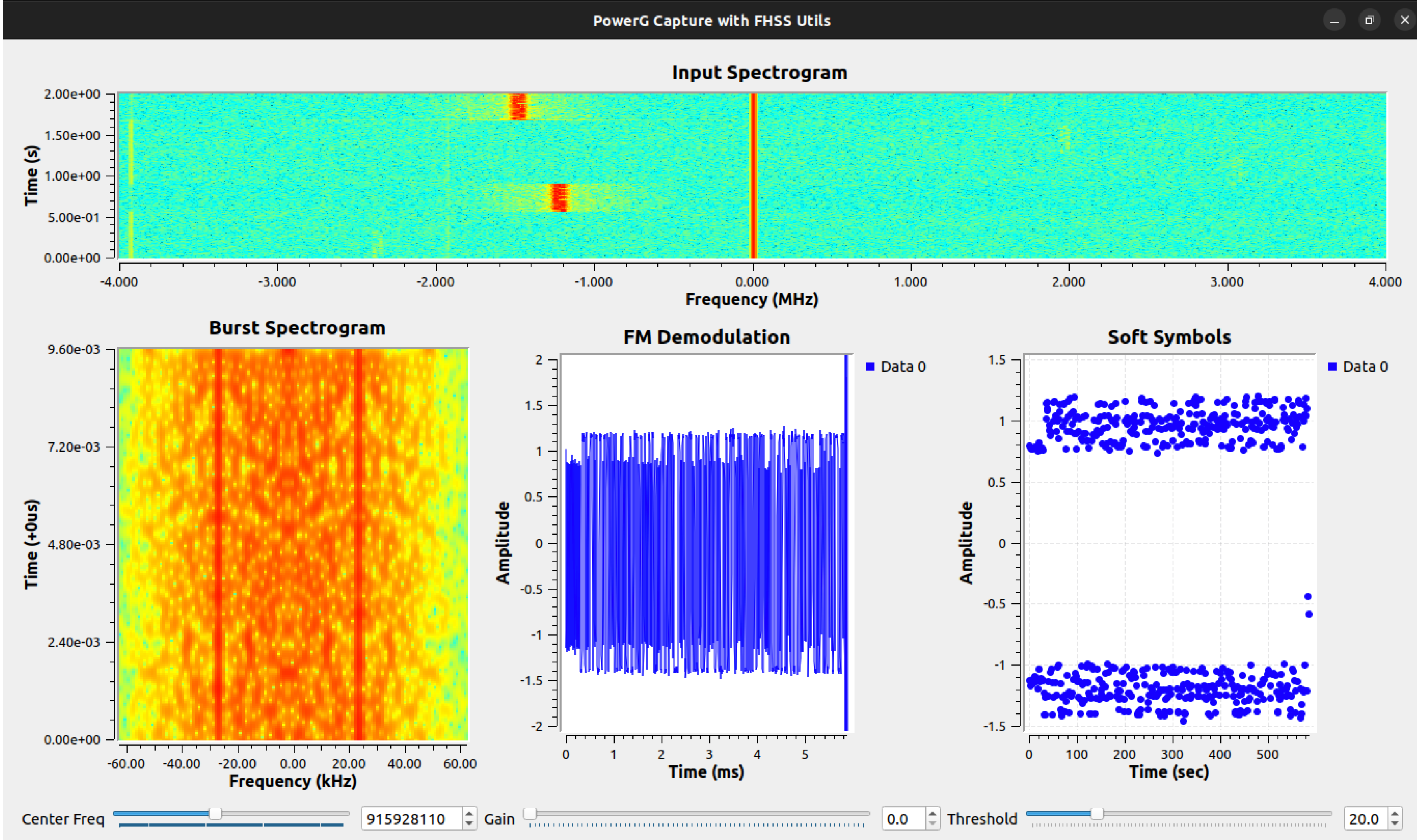


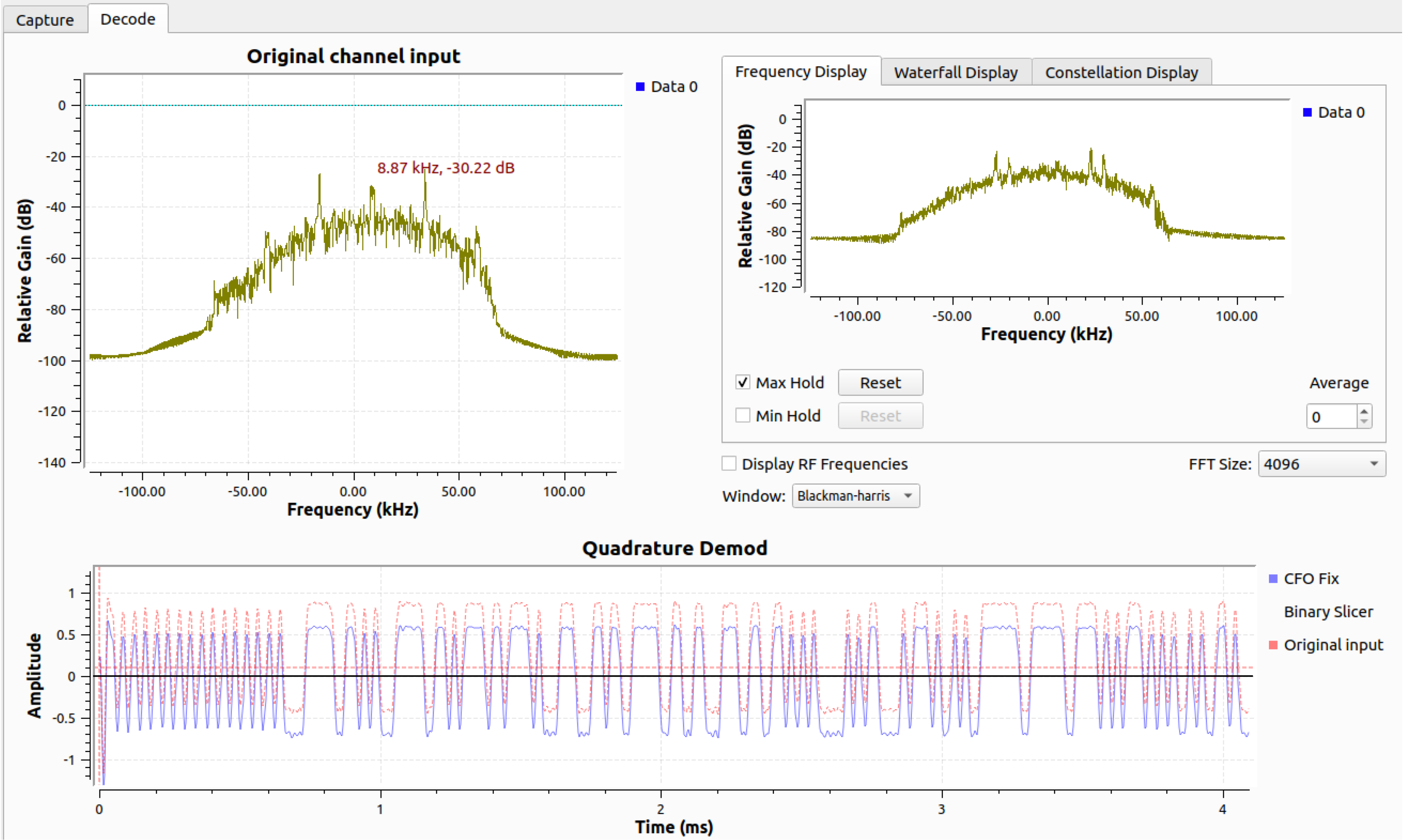
Firmware Host Communication Handling

- Host can configure and poll modem via UART, e.g.:
  - Set network keys
  - Perform factory reset
- Packets forwarded between network and host via UART
  - Some RF packets are handled directly by modem, e.g. pairing process
- Host handles higher-level things like:
  - configuration of network through control panel user interface
  - monitoring status of connected sensors
  - audible and visual alarms

Packet Capture with Frequency Hopping

- The channel hopping algorithm was initially unknown to us, and later found to be dependent on cryptographic keys
  - Difficult to capture with a single-channel sniffer
- The entire range of frequencies used by PowerG is less than 6.5 MHz
  - Possible to use SDR to capture all channels at once
- We used Sandia gr-fhss_utils for burst detection across the full frequency range to capture packets without needing to understand the channel hopping
- Using CFO correction to account for inaccuracies in burst detector centre frequency estimation and crystal inaccuracy
- Also does clock recovery (choose point on waveform to sample symbol)

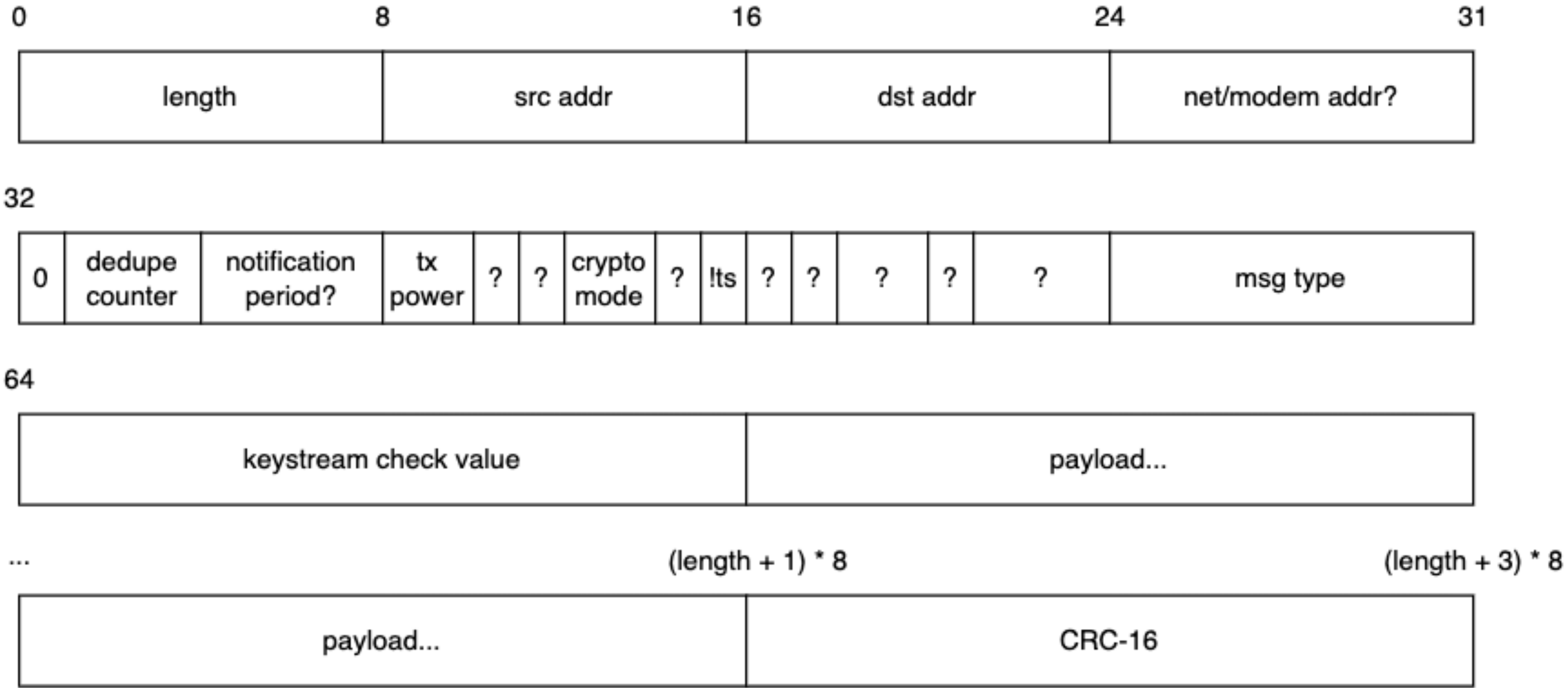

PowerG Capture with FHSS Utils
Input Spectrogram
Time (s)
Frequency (MHz)
Burst Spectrogram
Time (+0us)
Frequency (kHz)
FM Demodulation
Amplitude
Time (ms)
Data 0
Soft Symbols
Amplitude
Time (sec)
Data 0
Center Freq
915928110
Gain
0.0
Threshold
20.0


Capture
Decode
Original channel input
Data 0
8.87 kHz, -30.22 dB
Relative Gain (dB)
Frequency (kHz)
Frequency Display
Waterfall Display
Constellation Display
Max Hold
Reset
Min Hold
Reset
Average
0
Display RF Frequencies
FFT Size:
4096
Window:
Blackman-harris
Quadrature Demod
Amplitude
Time (ms)
CFO Fix
Binary Slicer
Original input

Packet Decoding

- GNU radio flow graph with Sandia gr-pdu_utils and gr-fhss_utils
  - Detect burst
  - Estimate center frequency
  - Filter to burst region
  - Feed through quadrature demodulation
  - Perform clock recovery
  - Check for sync word
  - Output group of bits when sync word is detected
- We added a CC1101 dewhitening block to simplify looking at the output bytes

Packet Structure



!ts = no timestamp appended

```
Packet on channel 13 (Burst center frequency:
914,450,087 Hz)
Start time: 14.351108294263273
CRC-16 (CC1101): 0xd073 GOOD

== TIMESTAMP ==
included timestamp: 4293920256  (4293920256
rounded to 1/64s)
unknown 2 bytes before timestamp: 2063 (0x080f)

== CHANNEL HOPPING ==
default channel: 15
hop config 0 channel: 19
hop config 1 channel: 44
hop config 2 channel: 13

== HEADER ==
Packet length: 63 (0x3e + 1)
Payload length: 52
Src addr: 0xfe
Dst addr: 0x01
??? addr: 0xfd

Bit field bytes:
4: 01110000 (0x70)
5: 01000000 (0x40)
6: 00000000 (0x00)
```

```
dedupe counter: 7
notification period?:   0
no time info:   0
byte 5 bit 1:   0
nonce mode:     0
Tx power:       1 -> 14 dBm
byte 6 bit 7:   0
byte 6 bit 6:   0
byte 6 bit 5-4: 0
byte 6 bit 3:   0
byte 6 bit 2-0: 0

RF message type: 0x76
Keystream head: ffff
Nonce/crypto mode: 0

== BODY ==
Payload:
00000000: 1C 77 E6 00 00 28 00 0B  01 03 2D 13 14 01 00 11  .w...(....-.....
00000010: 01 03 29 01 10 03 10 00  00 00 00 00 01 24 12 00  ..)..........$..
00000020: 22 61 00 00 00 02 07 00  70 24 35 04 61 00 0F 08  "a......p$5.a...
00000030: 00 06 F0 FF                                       ....
```

Pairing Process

- Communication unencrypted
  - Modem enters pairing mode
  - Device enters pairing mode, sends packets on static channel using source address 0xFE:
    - Type 0x76 device ad message to addresses 0x01-0x09
      - Includes device “long ID”
    - Sends type 0x70 to modem addr (01) to request pairing
  - Modem responds on static channel with type 0x71 message containing the network’s primary key (AES key)
- Switch to crypto mode 2 (encrypt with network key, all zero nonce, network clock)
  - Modem sends type 0x51 message containing its clock
  - Device sends type 0x72 message containing device info again
  - Modem sends type 0x73 with “secondary key” (configured 96 bit ‘nonce’)
- Switch to crypto mode 3 (encrypt with network key, configured nonce, network clock)
  - Modem sends 0x74 messages that appear to indicate when it exits pairing mode
  - Device is paired to network, starts sending its data (type 0x52)

Data messages (RF type 0x52)

- Using packet capture & decode capability, we can now do simple dynamic analysis tests to learn about payloads for specific peripherals
- Perform different actions in different captures, select unique packet payloads from each
- Cancel out any fields that appear to always change across all capture sets (likely some counter, timer, etc.)
- E.g., identifying messages for a door contact sensor (open, close, tampered):

```
tamper 1: 640001 1c77e6 05 03 fd 01 00 (??)
tamper 2: 640005 1c77e6 05 03 40 01 00 (??)
tamper 3: 040006 1c77e6 05 11 27 01 00 2801 00 1101 00 f1010926 036a 05 00 (also closed)
tamper 4: 040008 1c77e6 05 11 27 01 01 2801 00 1101 00 f1010926 036a 05 00 (also closed)
closed:   040006 1c77e6 05 11 27 01 00 2801 01 1101 00 f1010926 0381 05 00
open:     640008 1c77e6 05 11 27 01 00 2801 01 1101 01 f1010926 036b 05 53
                 |_dev ID  |_length        |_tamper |                  |
                                                    |_ opened/closed   |
                                                                       |_ also opened/closed?
```

Areas for Future Work - Jamming

- Despite the claims of FHSS preventing jamming, it should still be feasible
  - The band used is narrow enough for jamming all 50 channels simultaneously to be feasible at a reasonably low power level
  - Directional antennas can be pointed at the panel to minimize power requirements and disturbance of the surrounding RF environment
  - Per-channel reactive jamming can be implemented to only jam during packet transmissions
- PowerG modems have a jamming detection mechanism that reports jamming to the panel when RSSIs are repeatedly above a threshold
  - Can the the jamming detection be confused, perhaps through modulated jamming signals?

Areas for Future Work - Protocol Attack Surface

- Numerous message types exist, many with non-trivial structures
  - Additional message types for various sensors/sirens/locks/remotes still need to be studied
- All message parsing is done in C modem firmware and C++ panel code
  - Possibility of memory safety issues
  - Possibility of pivoting from compromising the PowerG network to compromising modem firmware to compromising host (panel) software

Disclosed Vulnerabilities

- Issues disclosed by NCC Group today:
  - Insecure Pairing Process in PowerG
  - AES-CTR Nonce Reuse in PowerG Packet Encryption
  - PowerG Packet Encryption Not Authenticated
- One vulnerability affecting popular PowerG products is still being withheld under an extended disclosure timeline to provide additional opportunity for Johnson Controls to address it.

Disclosure Timeline

- March 28, 2024: First contact attempt with productsecurity@jci.com, initial disclosure of three vulnerabilities
- April 10, 2024: Second contact attempt with productsecurity@jci.com, disclosing one additional vulnerability
- May 15th, 2024: Third, fourth, and fifth contact attempts through LinkedIn and customer service contacts
- June 10th, 2024: NCC Group discloses issues to CERT/CC and CISA
- June 13th, 2024: Vendor responds to acknowledge report, disagree with CVSS ratings
- June 18th, 2024: NCC Group maintains 90 day disclosure window for three vulnerabilities, volunteers to extend timeline for one vulnerability

Tools Used

- SigDigger – for radio signal visualization and baseband analysis
- Universal Radio Hacker (URH) – for black-box radio protocol analysis
- Ghidra – for firmware analysis
- Symgrate – for black-box symbol identification
- GNU Radio – for building packet capture pipeline
- Sandia National Laboratories GNU Radio Utilities:
    - gr-pdu_utils: demodulation, clock recovery, sync word detection
    - gr-fhss_utils: capturing frequency hopping spread spectrum signals (burst detection, frequency offset correction)

Question & Answers