7 Reverse Engineering the LoRa PHY

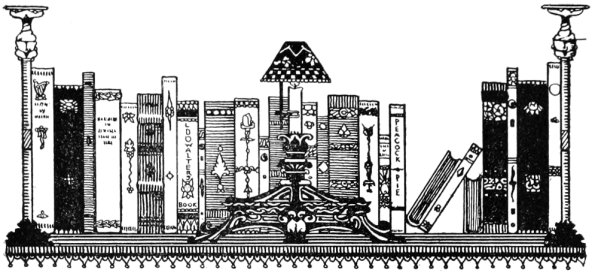
by Matt Knight

It's 2016, and everyone's favorite inescapable buzzword is IoT, or the "Internet of Things." The mere mention of this phrase draws myriad reactions, depending on who you ask. A marketing manager may wax philosophical about swarms of connected cars eradicating gridlock forever, or the inevitability of connected rat traps intelligently coordinating to eradicate vermin from midtown Manhattan,[18] while a security researcher may just grin and relish in the plethora of low-power stacks and new attack surfaces being applied to cyber-physical applications.

IoT is marketing speak for connected embedded devices. That is, inexpensive, low power, resource constrained computers that talk to each other, possibly on the capital-I Internet, to exchange data and command and control information. These devices are often installed in hard to reach places and can be expected to operate for years. Thus, easy to configure communication interfaces and extreme power efficiency are crucial design requirements. While 2G cellular has been a popular mechanism for connecting devices in scenarios where a PAN or wired technology will not cut it, AT&T's plans to sunset 2G on January 1, 2017 and LTE-M Rel 13's distance to widespread adoption presents an opportunity for new wireless specifications to seize market share.

LoRa is one such nascent wireless technology that is poised to capture this opportunity. It is a Low Power Wide Area Network (LPWAN), a class of wireless communication technology designed to connect low power embedded devices over long ranges. LoRa implements a proprietary PHY layer; therefore the details of its modulation are not published.

This paper presents a comprehensive blind signal analysis and resulting details of LoRa's PHY, chronicles the process and pitfalls encountered along the way, and aspires to offer insight that may assist security researchers as they approach their future unknowns.



7.1 Casing the Job

I first heard of LoRa in December 2015, when it and other LPWANs came up in conversation among neighbors. Collectively we were intrigued by its advertised performance and unusual modulation, thus I was motivated to track it down and learn more. In the following weeks, I occasionally scanned the 900 MHz ISM spectrum for signs of its distinctive waveform (more on that soon), however searches in the New York metropolitan area, Boston, and a colleague's search in San Francisco yielded no results.

Sometime later I found myself at an IoT security meetup in Cambridge, MA that featured representatives from Senet and SIGFOX, two major LPWAN players. Senet's foray into LoRa started when they sought to remotely monitor fluid levels in home heating oil tank measurement sensors to improve the existing process of sending a guy in a truck to read it manually. Senet soon realized that the value of this infrastructure extended far beyond the heating oil market and has expanded their scope to becoming a IoT cellular data carrier of sorts. While following up on the company I happened upon one of their marketing videos online. A brief segment featured a grainy shot of a coverage map, which revealed just enough to suggest the presence of active infrastructure in Portsmouth, NH. After quick drive with my Ettus B210 Software Defined Radio, I had my first LoRa captures.

7.2 First Observations and OSINT

LoRa's proprietary PHY uses a unique chirp spread spectrum (CSS) modulation scheme, which encodes information into RF features called chirps. A chirp

[18] LoRaWan in the IoT Industrial Panel, presentation by Jun Wen of Cisco.

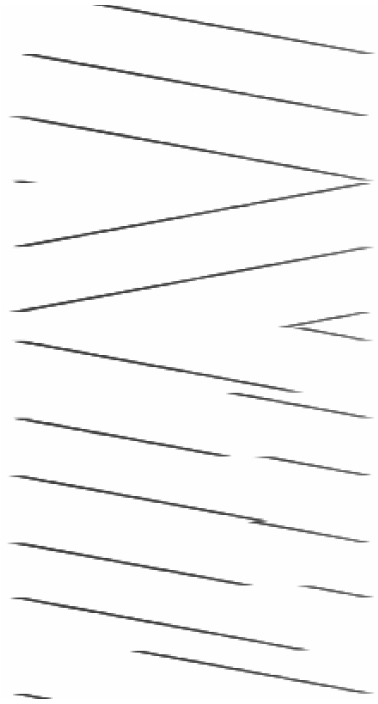


Figure 11. Spectrogram of a LoRa packet.

is a signal whose frequency is increasing or decreasing at a constant rate, and they are unmistakable within the waterfall. A chirp-based PHY is shown in Figure 11.

Contrasted with FSK or OFDM, two common PHYs, the differences are immediately apparent.

Modulation aside, visually inspecting a spectrogram of LoRa's distinct chirps reveals a PHY structure that is similar to essentially all other digital radio systems: the preamble, start of frame delimiter, and then the data or payload.

Since LoRa's PHY is proprietary, no PHY layer specifications or reference materials were available. However, thorough analysis of open source and readily available documentation can greatly abbreviate reverse engineering processes. When I conducted this investigation, a number of useful documents were available.

First, the Layer 2+ LoRaWAN stack is published, containing clues about the PHY.

Second, several application notes were available for Semtech's commercial LoRa modules.[19] These were not specs, but they did reference some PHY-layer components and definitions.

Third, a European patent filing from Semtech described a CSS modulation that could very well be LoRa.

Finally, neighbors who came before me had produced open-source prior art in the form of a partial `rtl-sdrangelove` implementation and a wiki page,[20] however in my experience the `rtl-sdrangelove` attempt was piecemeal and neglected and the wiki contained only high level observations. These were not enough to decode the packets that I had captured in New Hampshire.

7.3 Demodulation

OSINT gathering revealed a number of key definitions that informed the reverse engineering process. A crucial notion is that of the spreading factor (SF): the spreading factor represents the number of bits packed into each symbol. A symbol, for the unordained, is a discrete RF energy state that represents some quantity of modulated information (more on this later.) The LoRaWAN spec revealed that the chirp bandwidth, that is the width of the channel that the chirps traverse, is 125 kHz,

[19] Semtech AN1200.18, AN1200.22.

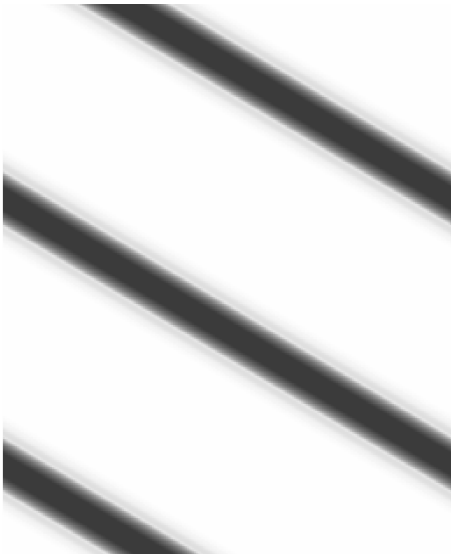
[20] Decoding LoRa on the RevSpace Wiki

250 kHz, or 500 kHz within American deployments. The chirp rate, which is intuitively the first derivative of the signal's frequency, is a function of the spreading factor and the bandwidth: it is defined as $\text{bandwidth}/2^{(\text{spreading\_factor})}$. Additionally, the absolute value of the downchirp rate is the same as the upchirp rate.[21]

Back to the crucial concept of symbols. In LoRa, symbols are modulated onto chirps by changing the instantaneous frequency of the signal – the first derivative of the frequency, the chirp rate, remains constant, while the signal itself "jumps" throughout its channel to represent data. The best way to intuitively think of this is that the modulation is frequency-modulating an underlying chirp. This is analogous to the signal alternating between two frequencies in a 2FSK system, where one frequency represents a `0` and the other represents a `1`. The underlying signal in that case is a signal of constant frequency, rather than a chirp, and the number of bits per symbol is 1. How many data bits are encoded into each frequency jump within LoRa? This is determined by the spreading factor.

The first step to extracting the symbols is to dechirp the received signal. This is done by channelizing the received signal to the chirp's bandwidth and multiplying the result against a locally-generated complex conjugate of whichever chirp is being extracted.

A locally generated chirp might look like this.

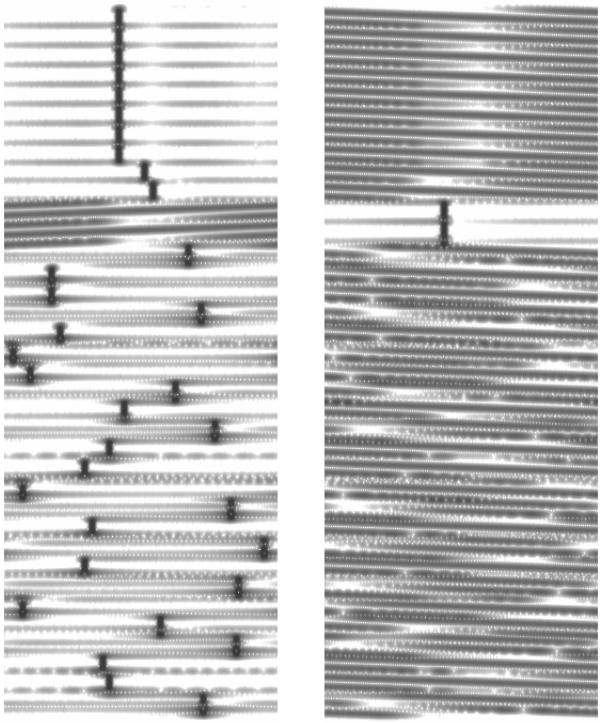


[21] See Semtech AN1200.22.

Since both upchirps and downchirps are present in the modulation, the signal should be multiplied against both a local upchirp and downchirp, which produces two separate IQ streams. Why this works can be reasoned intuitively, since waves obey superposition, multiplying a signal with frequency $f0$ against a signal with frequency $-f0$ results in a signal with frequency 0, or DC. If a chirp is multiplied against a copy of itself, it will result in a signal of $2 * f0$, which will spread its energy throughout the band. Thus, generating a local chirp at the negative chirp rate of whichever chirp is being processed

results in RF features with constant frequency that can be handled nicely.

In following examples, the left image shows de-chirped upchirps while the right shows de-chirped downchirps:



This de-chirped signal may be treated similarly to MFSK, where the number of possible frequencies is $M = 2^{(\text{spreading\_factor})}$. The Fast Fourier Transform (FFT) is the tool used to perform the actual symbol measurement. Fourier analysis shows that a signal can be modeled as a summed series of basic periodic functions (i.e., a sine wave) at various frequencies. A FFT decomposes a signal into the frequency components that comprise it, returning the power and phase of each component present. Each component to be extracted is colloquially called a "bin;" the number of bins is specified as the "FFT size" or "FFT width."

Thus, by taking an M-bin wide FFT of each IQ stream, the symbols may be resolved by finding the argmax, which is the bin with the most powerful component of each FFT. This works out nicely because a de-chirped CSS symbol turns into a signal with constant frequency; all of the symbol's energy should fall into a single bin.[22]

With the signal de-chirped, the remainder of the demodulation process can be described in three steps. These steps mimic the process required for essentially all digital radio receivers.

First, we'll identify the start of the packet by finding a preamble. Then, we'll synchronize with the start of the packet, so that we may conclude in demodulating the payload by measuring its aligned symbols.

7.3.1 Finding the Preamble

A preamble is a feature included in modulation schemes to announce that a packet is soon to follow. By visual inspection, we can infer that LoRa's preamble is represented by a series of continuous upchirps. Once de-chirped and passed through an FFT, all of the preamble's symbols wind up residing within the same FFT bin. Thus, a preamble is detected if enough consecutive FFTs have the same argmax.

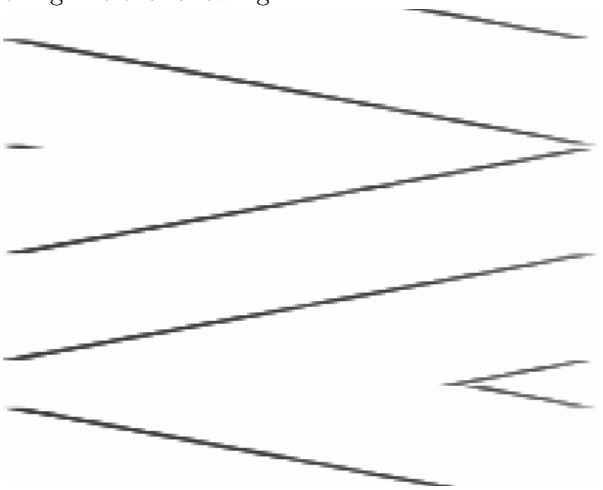
7.3.2 Synchronizing with the SFD

With our receiver aware that it's about to receive a packet, the next step is to accurately synchronize with it so that symbols can be resolved accurately. To facilitate this, modern radio systems often advertise the start of the packet's data unit with a Start of

[22]It may be possible to do this using FM demodulation rather than FFTs, however using FFTs preserves power information that is useful for framing the packet without knowing its definitive length.

Frame Delimiter, or SFD, which is a known symbol distinct from the preamble that receivers are programmed to look for. For LoRa, this is where the downchirps come in.

The SFD is composed of two and one quarter downchirps, while all the other symbols are represented by upchirps. With preamble having been found, our receiver should look for two consecutive downchirps to synchronize against. It looks something like the following:



Accurate synchronization is crucial to properly resolving symbols. If synchronization is off by enough samples, when FFTs are taken each symbol's energy will be divided between two adjacent FFTs. Until now, the FFT process used to resolve the symbols processed $2^{(\text{spreading\_factor})}$ samples per FFT with each sample being processed exactly once, however after a few trial runs it became evident that this coarse synchronization would not be sufficiently accurate to guarantee good fidelity.

Increasing the time-based FFT resolution was found to be a reliable method for achieving an accurate sync. This is done by shifting the stream of de-chirped samples through the FFT input buffer, processing each sample multiple times, to "overlap" adjacent FFTs. This increases the time-based resolution of the FFT process at the expense of being more computationally intensive. Thus, overlapping FFTs are only used to frame the SFD; non-overlapped FFTs with each sample being processed exactly once are taken otherwise to balance accuracy and computational requirements.

Technically there's also a sync word that precedes the SFD, but my demodulation process described in this article does not rely on it.

[23]European Patent #13154071.8/EP20130154071

7.3.3 Demodulating the Payload

Now synchronized against the SFD, we are able to efficiently demodulate the symbols in the payload by using the original non-overlapping FFT method. However, since our receiver's locally generated chirps are likely out of phase with the chirp used by the transmitter, the symbols appear offset within the set range $[0 : 2^{(\text{spreading\_factor})} - 1]$ by some constant. It was surmised that the preamble would be a reliable element to represent symbol `0`, especially given that the aforementioned sync word's value is always referenced from the preamble. A simple modulo operation to normalize the symbol value relative to the preamble's zero-valued bin produces the true value of the symbols, and the demodulation process is complete.

7.4 Decoding, and its Pitfalls

Overall, demodulation proved to not be too difficult, especially when you have someone like Balint Seeber feeding you advice and sagely wisdom. However, decoding is where the fun (and uncertainty) really began.

First, why encode data? In order to increase over the air resiliency, data is encoded before it is sent. Thus, the received symbols must be decoded in order to extract the data they represent.

The documentation I was able to gather on LoRa certainly suggested that figuring out the decoding would be a snap. The patent application describing a LoRa-like modulation described four decoding steps that were likely present. Between the patent and some of Semtech's reference designs, there were documented algorithms or detailed descriptions of every step. However, these documents slowly proved to be lies, and my optimism proved to be misplaced.

7.4.1 OSINT Revisited

Perhaps the richest source of overall hints was Semtech's European patent application.[23] The patent describes a CSS-based modulation with an uncanny resemblance to LoRa, and goes so far as to walk step-by-step through the encoding elements present in the PHY. From the encoder's perspective, the patent describes an encoding pipeline of forward error correction, a diagonal interleaver, data whitening, and gray indexing, followed by the just-described modulation process. The reverse process

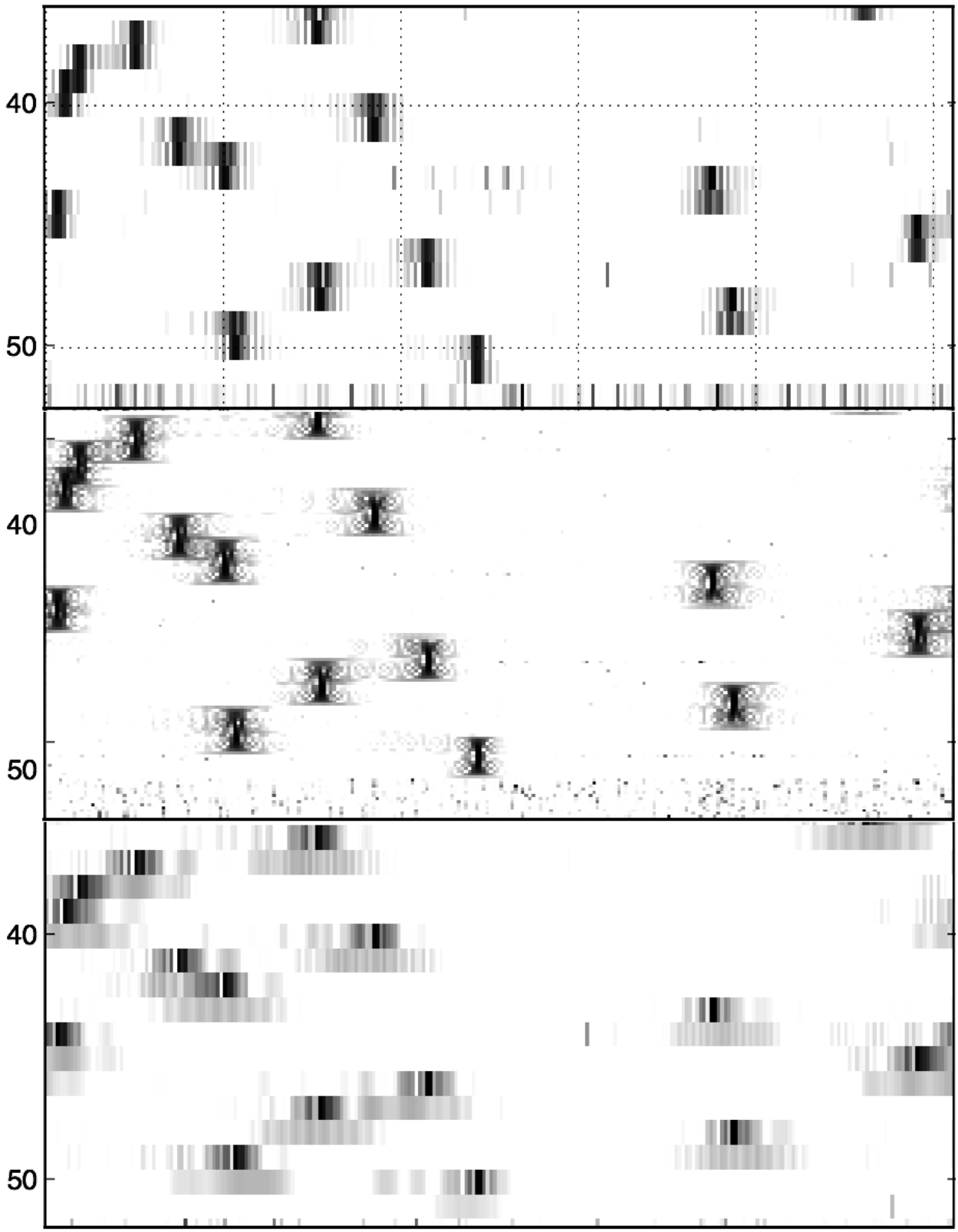


Figure 12. The top is pre-sync and non-overlapped, middle is pre-sync overlapped, bottom is synchronized and non-overlapped.

would be performed by the decoder. The patent even defines an interleaver algorithm, and Semtech documentation includes several candidate whitening algorithms.

The first thing to try, of course, was to implement a decoder exactly as described in the documentation. This involved, in order:

1. Undoing gray coding applied to the symbols.
2. Dewhitening using the algorithms defined in Semtech's documentation.
3. Deinterleaving using the algorithm defined in Semtech's patent.
4. Processing the Hamming forward error correction hinted at in Semtech's documentation.

First, let's review what we have learned about each step listed above based on open-source research, and what would be attempted as a result.

Gray Indexing Given the nomenclature ambiguity in the Semtech patent, I also decided to test no gray coding and reverse gray coding in addition to forward gray coding. These were done using standard algorithms.

Data Whitening Data whitening was a colossal question mark while looking at the system. An ideal whitening algorithm is pseudorandom, thus an effective obfuscator for all following components of the system. Luckily, Semtech appeared to have published the algorithm candidates in Application Note AN1200.18. Entitled "Implementing Data Whitening and CRC Calculation in Software on SX12xx Devices," it describes three different whitening algorithms that were relevant to the Semtech SX12xx-series wireless transceiver ICs, some of which support LoRa. The whitening document provided one CCITT whitening sequences and two IBM methods in C++. As with the gray indexing uncertainty, all three were implemented and permuted.

Interleaver Interleaving refers to methods of deterministically scrambling bits within a packet. It improves the effectiveness of Forward Error Correction, and will be elaborated on later in this text. The Semtech patent application defined a diagonal interleaver as LoRa's probable interleaver. It is a block-style non-additive diagonal interleaver that shuffles bits within a block of a fixed size. The interleaver is defined as: Symbol$(j, (i+j)\%\text{PPM}) =$ Codeword(i, j) where $0 <= i <$ PPM, $0 <= j < 4 +$ RDD In this case, PPM is set to the spreading factor (or $spreading\_factor - 2$ for the PHY header and when in low data rate modes), and RDD is set to the number of parity bits used by the Forward Error Correction scheme (ranging $[1:4]$).

There was only one candidate illustrated here, so no iteration was necessary.

Forward Error Correction The Semtech patent application suggests that Hamming FEC be used. Other documentation appeared to confirm this. A custom FEC decoder was implemented that originally just extracted the data bits from their standard positions within `Hamming(8,4)` codewords, but early results were negative, so this was extended to apply the parity bits to repair errors.

Using a Microchip RN2903 LoRa Mote, a transmitter that was understood to be able to produce raw frames, a known payload was sent and decoded using this process. However, the output that resulted bore no resemblance to the expected payload. The next step was to inspect and validate each of the algorithms derived from documentation.

After validating each component, attempting every permutation of supplied algorithms, and inspecting the produced binary data, I concluded that something in LoRa's described encoding sequence was not as advertised.

7.5 Taking Nothing for Granted

The nature of analyzing systems like this is that beneath a certain point they become a black box. Data goes in, some math gets done, RF happens, said math gets undone, and data comes out. Simple enough, but when encapsulated as a totality it becomes difficult to isolate and chase down bugs in each component. Thus, the place to start was at the top.

"Fred understands what they're saying since he converted to single sideband"

7.5.1 How to Bound a Problem

The Semtech patent describes the first stage of decoding as "gray indexing." Gray coding is a process that maps bits in such a way that makes it resilient to off-by-one errors. Thus, if a symbol were to be measured within ± 1 index of the correct bin, the gray coding would naturally correct the error. "Gray indexing," ambiguously referring to either gray coding or its inverse process, was initially understood to mean forward gray coding.

The whitening sequence was next in line. Data whitening is a process applied to transmitted data to induce randomness into it. To whiten data, the data is XORed against a pseudorandom string that is known to both the transmitter and the receiver. This does good things from an RF perspective, since it induces lots of features and transitions for a receiver to perform clock recovery against. This is functionally analogous to line coding schemes such as Manchester encoding, but whitening offers one pro and one con relative to line coding: data whitening does not impact the effective bit rate as Manchester encoding does,[24] but this comes at the expense of legibility due to the pseudorandom string.

At this point, it is important to address some of the assumptions and inferences that were made to frame the following approach. While the four decoding stages were thrown into question by virtue of the fact that at least one of the well-described algorithms was not correct, certain implied properties could be generalized for each class of algorithm, even if the implementation did not match exactly.

I made a number of assumptions at this point, which I'll describe in turn.

First, the interleaver in use is non-additive. This means that while it will reorder the bits within each interleaving block, it will not cause any additional bits to be set or unset. This was a reasonable assumption because many block-based interleavers are non-additive, and the interleaver defined in the patent is non-additive as well. Even if the interleaver used a different algorithm, such as a standard block interleaver or a different type of diagonal interleaver, it could still fit within this model.

Second, the forward error correction in use is Hamming FEC, with 4 data bits and 1-4 parity bits per codeword. FEC can be thought of as supercharged parity bits. A single parity bit can indicate the presence of an error, but if you use enough of them they can collectively identify and correct errors in place, without re-transmission. Hamming is specifically called out by the European patent, and the code rate parameter referenced throughout reference designs fits nicely within this model. The use of Hamming codes, as opposed to some other FEC or a cyclic coding scheme, was fortuitous because of a property of the Hamming code words. Hamming codeword mapping is deterministic based on the nybble that is being encoded. Four bits of data provide 16 possible codewords. When looking at `Hamming(8,4)` (which is the inferred FEC for LoRa code rate 4/8), 14 of the 16 codewords contain four set bits (`1`s) and four unset bits (`0`s). However, the code words for `0b0000` and `0b1111` are `0b00000000` and `0b11111111`, respectively.

Thus, following on these two assumptions, if a payload containing all `0x00`s or `0xFF`s were sent, then the interleaving and forward error correction should cancel out and not affect the output at all. This *reduces our unknown stages* in the decoding chain from four to just two, with the unknowns being gray indexing and whitening, and once those are resolved then the remaining two can be solved for!

Since "gray indexing" likely refers to gray coding, reverse gray coding, or no coding should it be omitted, this leaves only three permutations to try while solving for the data whitening sequence.

The first step was to take a critical look at the data whitening algorithms provided by Semtech AN1200.18. Given the detail and granularity in which they are described, plus the relevance of having come straight from a LoRa transceiver datasheet, it was almost a given that one of the three algorithms would be the solution. With the interleaver and FEC effectively zeroed out, and "gray indexing" reduced to three possible states, it became possible to test each of the whitening algorithms.

Testing each whitening algorithm was fairly

[24] Manchester's effective bit rate is 1/2 baud rate.

straightforward. A known payload of all `0x00`s or `0xFF`s (to cancel out interleaving and FEC) was transmitted from the Microchip LoRa Technology Mote and then decoded using each whitening algorithm and each of the possible "gray indexing" states. This resulted in 9 total permutations. A visual diff of the decoded data versus the expected payload resulted in no close matches. This was replaced with a diff script with a configurable tolerance for bits that did not match. This also resulted in no matches as well. One final thought was to forward compute the whitening algorithms in case there was a static offset or seed warm-up, as can be the case with other PRNG algorithms. Likewise, this did not reveal any close matches. This meant that either none of the given whitening algorithms in the documentation were utilized, or the assumptions that I made about the interleaver and FEC were not correct.

After writing off the provided whitening algorithms as fiction, the next course of action was to attempt to derive the real whitening algorithm from the LoRa transmitter itself. This approach was based on the previous observations about the FEC and interleaver and a fundamental understanding of how data whitening works. In essence, whitening is as simple as XORing a payload against a static pseudorandom string, with the same string used by both the transmitter and receiver. Since anything XORed with zero is itself, passing in a string of zeroes causes the transmitter to reveal a "gray indexed" version of its whitening sequence.

This payload was received, then transformed into three different versions of itself: one gray-coded, one unmodified, and one reverse gray-coded. All three were then tested by transmitting a set of `0xF` data nybbles and using each of the three "gray indexing" candidates and received whitening sequence to decode the payload. The gray coded and unmodified versions proved to be incorrect, but the reverse gray coding version successfully produced the transmitted nybbles, and thus in one fell swoop, I was able to both derive the whitening sequence and discern that "gray indexing" actually referred to the reverse gray coding operation. With "gray indexing" and whitening solved, I could turn my attention to the biggest challenge: the interleaver.

7.5.2 The Interleaver

At this point we've resolved two of the four signal processing stages, disproving their documentation in the process. Following on this, the validity of the interleaver definition provided in Semtech's patent was immediately called into question.

A quick test was conducted against a local implementation of said interleaver: a payload comprised of a repeated data byte that would produce a `Hamming(8,4)` codeword with four set and four unset bits was transmitted and the de-interleaved frame was inspected for signs of the expected codeword. A few other iterations were attempted, including reversing the diagonal offset mapping pattern described by the patent and using the inverse of the algorithm (i.e., interleaving the received payload rather than de-interleaving it). Indeed, I was able to conclude that the interleaver implemented by the protocol is not the one suggested by the patent. The next logical step is to attempt to reverse it.

Within a transmitter, interleaving is often applied after forward error correction in order to make the packet more resilient to burst interference. Interleaving scrambles the FEC-encoded bits throughout the packet so that if interference occurs it is more likely to damage one bit from many codewords rather than several bits from a single codeword. The former error scenario would be recoverable through FEC, the latter would result in unrecoverable data corruption.

Block-based interleavers, like the one described in the patent, are functionally straightforward. The interleaver itself can be thought of as a two-dimensional array, where each row is as wide as the number of bits in each FEC codeword and the number of columns corresponds to the number of FEC codewords in each interleaver block. The data is then written in row-wise and read out column-wise; thus the first output "codeword" is comprised of the LSB (or MSB) of each FEC codeword. A diagonal interleaver, as suggested in the patent, offsets the column of the bit being read out as rows are traversed.

Understanding the aforementioned fundamentals of what the interleaver was likely doing was essential to approaching this challenge. Ultimately, given that a row-column or row-diagonal relationship defines most block-based interleavers, I anticipated that patterns that could be revealed if approached appropriately. Payloads were therefore constructed to reveal the relationship of each row or codeword with a corresponding diagonal or column. In order to reveal said mapping, the `Hamming(8,4)` codeword for `0xF` was leveraged, since it would fill each row

0x0000000F	0x000000F0	0x00000F00	0x0000F000	0x000F0000	0x00F00000	0x0F000000	0xF0000000
00100011	11000000	00001001	11010000	00000011	01000100	01000001	00001000
00010011	00100101	00000111	00001001	00000011	00000011	10000010	01000101
00001001	00010001	00000011	00000101	01000001	00000000	00100001	10000011
00000111	00001101	00000011	00000110	10000010	01000101	00010010	00100011
00000000	00001100	01000010	00001000	00100010	10001001	00001010	00010011
00000100	00000000	10000001	01000010	00010001	00100010	00000111	00001011
01000011	00000001	00100001	10000000	00001001	00010000	00000011	00000111
10000101	01000111	00010000	00100101	00000000	00001111	00000101	00000111

Figure 13. Symbol Tests

with eight contiguous bits at a time. Payloads consisting of seven `0x0` codewords and one `0xF` codeword were generated, with the nybble position of `0xF` iterating through the payload. See Figure 13.

As one can see, by visualizing the results as they would be generated by the block, patterns associated with each codeword's diagonal mapping can be identified. The diagonals are arbitrarily offset from the corresponding row/codeword position. One important oddity to note is that the most significant bits of each diagonal are flipped.

While we now know how FEC codewords map into block diagonals, we do not know where each codeword starts and ends within the diagonals, or how its bits are mapped. The next step is to map the bit positions of each interleaver diagonal. This is done by transmitting a known payload comprised of FEC codewords with 4 set and 4 unset bits and looking for patterns within the expected diagonal.

```
Payload: 0xDEADBEEF
bit 76543210
     00110011
     10111110
     11111010
     11011101
     10000010
     10000111
     11000000
     10000010
```

Reading out the mapped diagonals results in the following table.

	T							Bot
D	1	0	1	0	0	0	0	1
E	0	1	1	1	0	1	0	0
A	0	1	0	1	1	0	0	0
D	1	0	1	1	0	0	0	0
B	1	1	0	0	0	0	1	0
E	0	1	1	1	0	1	0	0
E	0	1	1	1	0	1	0	0
F	1	1	1	1	1	1	1	1

While no matches immediately leap off the page, manipulating and shuffling through the data begins to reveal patterns. First, reverse the bit order of the extracted codewords:

	B							Top
D	1	0	0	0	0	1	0	1
E	0	0	1	0	1	1	1	0
A	0	0	0	1	1	0	1	0
D	0	0	0	0	1	1	0	1
B	0	1	0	0	0	0	1	1
E	0	0	1	0	1	1	1	0
E	0	0	1	0	1	1	1	0
F	1	1	1	1	1	1	1	1

And then have a look at the last nybble for each of the highlighted codewords:

	B							Top
D	1	0	0	0	0	1	0	1
E	0	0	1	0	**1**	**1**	**1**	**0**
A	0	0	0	1	**1**	**0**	**1**	**0**
D	0	0	0	0	**1**	**1**	**0**	**1**
B	0	1	0	0	0	0	1	1
E	0	0	1	0	**1**	**1**	**1**	**0**
E	0	0	1	0	**1**	**1**	**1**	**0**
F	1	1	1	1	**1**	**1**	**1**	**1**

Six of the eight diagonals resemble the data embedded into each of the expected FEC encoded codewords! As for the first and fifth codewords, it is possible they were damaged during transmission, or that the derived whitening sequence used for those positions is not exact. That is where FEC proves its mettle – applying `Hamming(8,4)` FEC would repair any single bit errors that occurred in transmission. The Hamming parity bits that are expected with each codeword are calculated using the Hamming FEC algorithm, or can be looked up for standard schemes like `Hamming(7,4)` or `Hamming(8,4)`.

```
     Data(8,4) Parity Bits
0xD  1101      1000
0xE  1110      0001
0xA  1010      1010
0xD  1101      1000
0xB  1011      0100
0xE  1110      0001
0xE  1110      0001
0xF  1111      1111
```

While the most standard `Hamming(8,4)` bit order is: p1, p2, d1, p3, d2, d3, d4, p4 (where p are parity bits and d are data bits), after recognizing the above data values we can infer that the parity bits are in a nonstandard order. Looking at the diagonal codeword table and the expected `Hamming(8,4)` encodings together, we can map the actual bit positions:

	Bot							Top
	p1	p2	p4	p3	d1	d2	d3	d4
D	1	0	0	0	0	1	0	1
E	0	0	1	0	1	1	1	0
A	0	0	0	1	1	0	1	0
D	0	0	0	0	1	1	0	1
B	0	1	0	0	0	0	1	1
E	0	0	1	0	1	1	1	0
E	0	0	1	0	1	1	1	0
F	1	1	1	1	1	1	1	1

Note that parity bits three and four are swapped. With that resolved, we can use the parity bits to decode the forward error correction, resulting in four bits being corrected, as shown in Figure 14.

That's LoRa!

– — — – — — — — – — –

Having reversed the protocol, it is important to look back and reflect on how and why this worked. As it turned out, being able to make assumptions and inferences about certain goings-on was crucial for bounding the problem and iteratively verifying components and solving for unknowns. Recall that by effectively canceling out interleaving and forward error correction, I was able to effectively split the problem in two. This enabled me to solve for whitening, even though "gray indexing" was unknown there were only three permutations, and with that in hand, I was able to solve for the interleaver, since FEC was understood to some extent. Just like algebra or any other scientific inquiry, it comes down to controlling your variables. By stepping through the problem methodically and making the right inferences, we were able to reduce 4 independent variables to 1, solve for it, and then plug that back in and solve for the rest.

7.6 Remaining Work

While the aforementioned process represents a comprehensive description of the PHY, there are a few pieces that will be filled in over time.

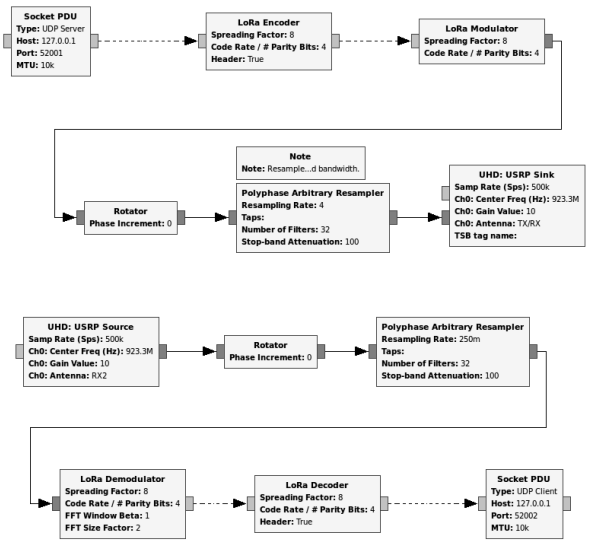
The LoRa PHY contains an optional header with its own checksum. I have not yet reversed the header, and the Microchip LoRa module I've used to generate LoRa traffic does not expose the option of disabling the header. Thus I cannot zero those bits out to calculate the whitening sequence applied to it. It should be straightforward to fill in with the correct hardware in hand.

The PHY header and service data unit/payload CRCs have not been investigated for the same reason. This should be easy to resolve through the use of a tool like CRC RevEng once the header is known.

In my experience, for demodulation purposes clock recovery has not been necessary beyond getting an accurate initial sync on the SFD. However should clock drift pose a problem, for example if transmitting longer messages or using higher spreading factors which have slower data rates/longer over-the-air transmission times, clock recovery may be desirable.

7.7 Shameless Plug

I recently published an open source GNU Radio OOT module that implements a transceiver based on this derived version of the LoRa PHY. It is presented to empower RF and security researchers to investigate this nascent protocol.[25]



[25] `git clone https://github.com/BastilleResearch/gr-lora`
`unzip pocorgtfo13.pdf gr-lora.tar.bz2`

									Top
	p1	p2	p4	p3	d1	d2	d3	d4	
D	1	0	0	0	**1**	1	0	1	1101 = 0xD
E	0	0	1	0	1	1	1	0	1110 = 0xE
A	**1**	0	0	1	1	0	1	0	1010 = 0xA
D	**1**	0	0	0	1	1	0	1	1101 = 0xD
B	0	1	0	0	**1**	0	1	1	1011 = 0xB
E	0	0	1	0	1	1	1	0	1110 = 0xE
E	0	0	1	0	1	1	1	0	1110 = 0xE
F	1	1	1	1	1	1	1	1	1111 = 0xF

Figure 14. Forward Error Corrected bits shown in bold

7.8 Conclusions and Key Takeaways

Presented here is the process that resulted in a comprehensive deconstruction of the LoRa PHY layer, and the details one would need to implement the protocol. Beyond that, however, is a testament to the challenges posed by red herrings (or three of them, all at once) encountered throughout the reverse engineering process. While open source intelligence and documentation can be a boon to researchers – and make no mistake, it was enormously helpful in debunking LoRa – one must remember that even the most authentic sources may sometimes lie!

Another point to take away from this is the importance of bounding problems as you solve them, including through making informed inferences in the absence of perfect information. This of course must be balanced with the first point about OSINT, is knowing when to walk away from a source. However as illustrated above, drawing appropriate conclusions proved integral to reducing and solving for each of the decoding elements within a black-box methodology.

The final thought I will leave you with is that wireless doesn't just mean Wi-Fi anymore - it includes cellular, PANs, LPWANs, and everything in between. Accordingly, a friendly reminder that security monitoring and test tools don't exist until someone creates them. Monitor mode and Wireshark weren't always a thing, so don't take them for granted: it's time to make the next generation of wireless networks visible to researchers, because know it or not it is already here and is here to stay.