Preface

I think it's safe to say that nobody really likes RFID drink control. Looking online you'll find countless Reddit posts or articles complaining about how Disney or Universal or their University is using RFID enabled drinkware to limit refills. Many people have speculated on how to defeat it. Some tried using their Flipper but to no avail, some claimed the data was stored off tag, on a server, and some even said it was impossible. In fact, I can find no evidence online that anybody has ever successfully hacked a Validfill tag. Well I'm here to tell you most of those people are wrong. It is possible, but it's a lot harder than you might think.

Considerations (found through research or testing)

- Data is stored directly on the tag rather than in the cloud/on a server
- Validfill uses UHF rfid technology (ISO-18000-6c, EPC c1g2, 902MHz - 927MHz) rather than nfc (13.56MHz, FlipperZero cannot read UHF without an expansion card)
- The EPC c1g2 spec allows for password protection on the tag. Validfill uses a write password meaning the tag can be read in any state but only written to with the correct password
- Similarly, using an unprotected tag with a Coke Freestyle machine also fails due to passwords not matching, so even if we clone the tag, the password must be correct in order for a successful scan
- Brute forcing the password is possible by attempting a read or write operation repeatedly, but the password is 32bit and the average read/write takes 50us meaning to test all passwords it would take nearly 10 years
- For the reader to unlock the tag, the password MUST be sent at some point to the tag which could allow for an eavesdropping attack via an rf receiver such as an SDR (software defined radio)
- The EPC c1g2 spec requires the sent password to be encrypted with a random number generated by the tag. This means we must intercept both the reader to tag AND tag to reader communication to reconstruct the password
- Reader to tag communication is very strong and can be received from meters away but the tag to reader communication uses backscatter, meaning its much lower amplitude relative to the reader signal and thus much harder to receive
- This means to receive this signal, the antenna must be very close to the tag as well as be positioned very carefully relative to the tag's antenna to maximize signal quality
- FCC regulations also require all readers to "frequency hop" meaning transmissions can and will occur at random frequencies from 902MHz to 927MHz. This means to improve our chances of receiving a quality transmission, we'll need to have a high sampling bandwidth
- After decoding the password we can finally use said password to write new data to the tag giving us full control of all the parameters of the cup
- The encoding of data on the tag is not public but using a data from a new tag, say with 100 fills, we can write said data to an old/alternate tag to load it with 100 fills as well
- It's possible that ValidFill uses the same password for all their cups, which I doubt, but it's possible and I don't want the heat. For this reason, I won't be releasing the full password that I decoded. In addition, the signals I decoded, as well as the EPCs used are example data and were not recovered from real tags.

Pre-Procedure

Readings

The second page of the Practical Eavesdropping of control data from EPC Gen2 article, figure 1 and the associated reading provides a great diagram and explanation of the EPC c1g2 protocol.

The official standard for the EPC c1g2 protocol gives very important info on reader and tag signal encoding in **sections 6.3.1.2 and 6.3.1.3** as well as command structure which is covered in **section 6.3.2.11.3**.

Equipment

An EPC c1g2 compliant UHF reader like this PISwords ISO18000 reader from AliExpress. This will allow us to read and write to the tag.

A cheap software defined radio such as the RTL-SDR from Amazon as well as a 902MHz to 927MHz antenna. This will allow us to eavesdrop on tag communication. Also recommended would be a USB extension cable so that you can freely move the radio.

Software

The PISwords reader comes with a demo program that we will use to read and write to the tag. The program is called **UHFReader09demomain.exe** can be found at "\Piswordsuhfreadernew\Piswordsuhfreader\USBreaderSoftware\Demo\C#\UHFReader09demomain\bin\x86\Debug"

We will be using Osmocom in conjunction with the RTL-SDR to capture and record reader and tag communication.

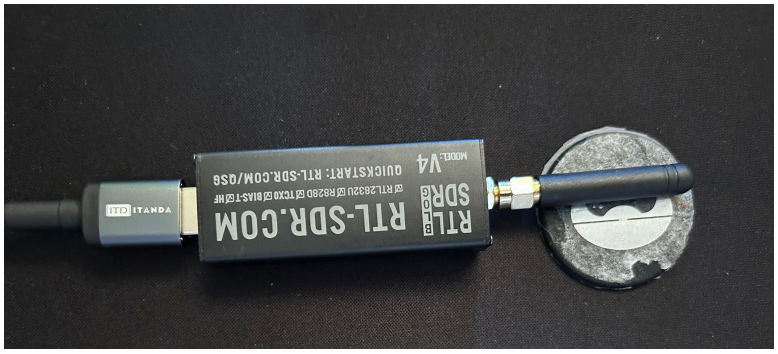
We will also be using the Inspectrum tool to analyze the resulting file and convert the FFT graph into an amplitude graph.

Signal decoding can be done visually using a drawing app such as tldraw we'll use to block out each of the bits.

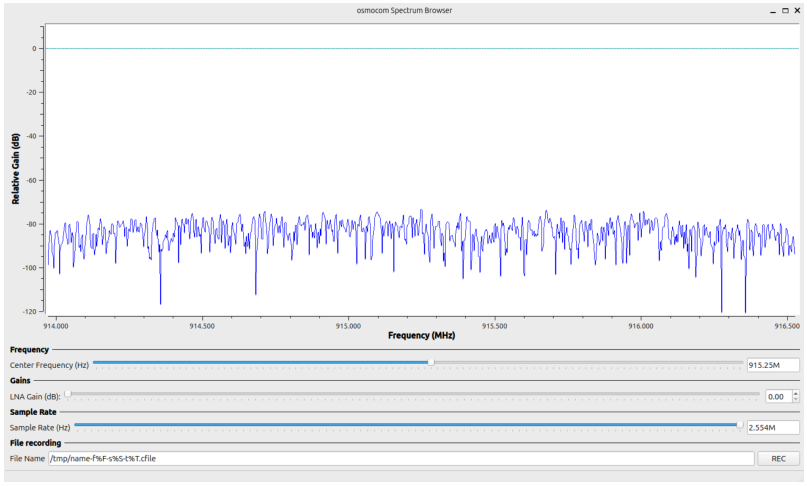
Procedure

Gathering Data

The first step in getting quality data is assembling the ideal setup. From testing I found that positioning the SDR's antenna directly in the middle of the tag's coil and parallel to the slit. Additionally, testing showed that the tag should be in direct contact with the reader on the Freestyle machine. Testing also showed that the Freestyle machine also seems to have a sensor for the bottle itself in addition to the tag so a bottle should be placed on top of the setup when scanning. You should leave the tag on the reader for at least 20-30 seconds to give you more data to choose from.

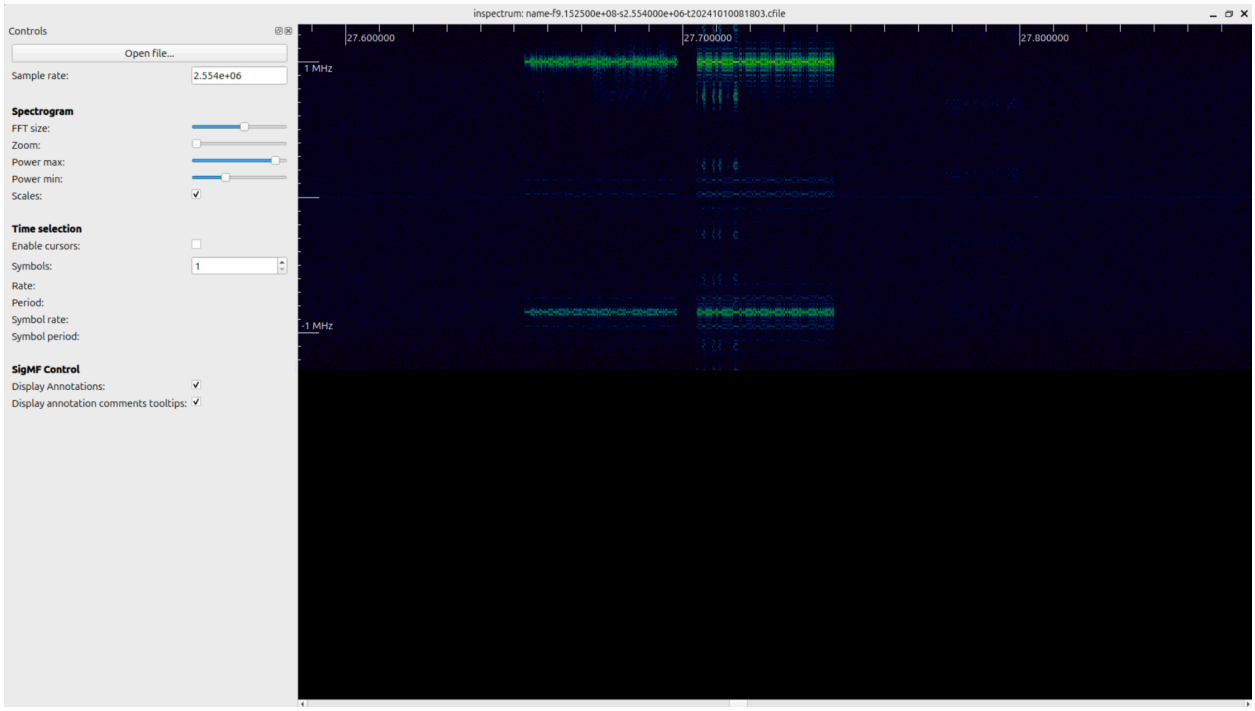


Using Osmocom, set the center frequency around 910MHz - 915MHz, testing showed this range was able to receive enough transmissions regardless of frequency hopping. Also, set the sample rate to the max, in my case, 2.55MHz. The higher the sample rate, the greater the bandwidth, and the more likely you are to receive a transmission.

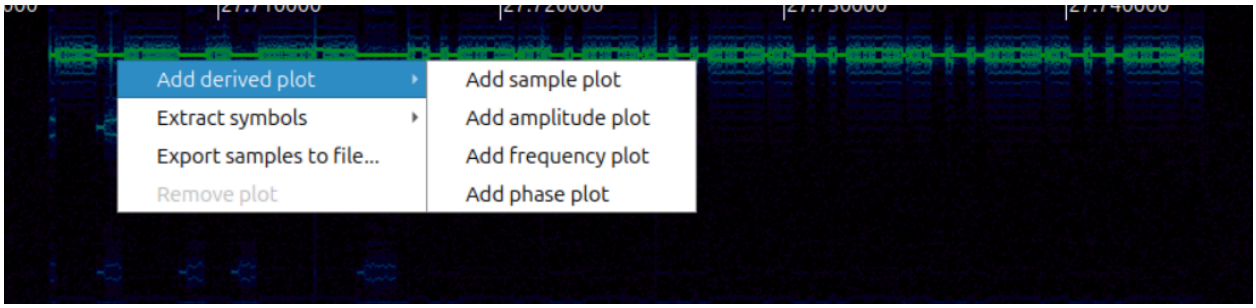


Analyzing the Transmission

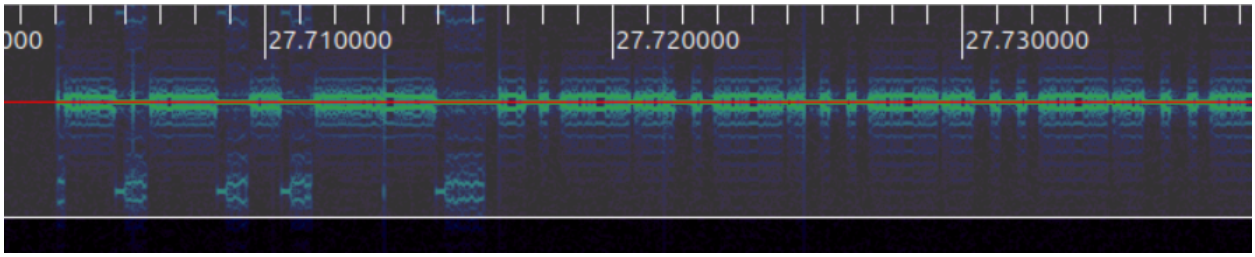
Using Inspectrum, import the file that Osmocom made. Set the **zoom** slider to min and drag the **FFT size** slider until it fills most of the screen. Leave the **power max** slider at max and adjust the **power min** slider until the background of the display is a dark blue. Start scrolling until you find a bright bar, that's a transmission.



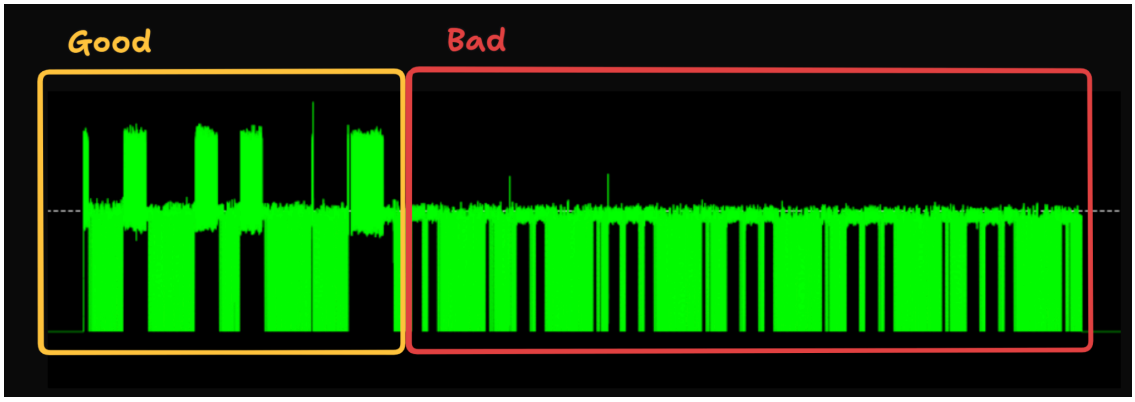
Right click on the transmission you want and select the **derived plot** and then **add amplitude plot**.



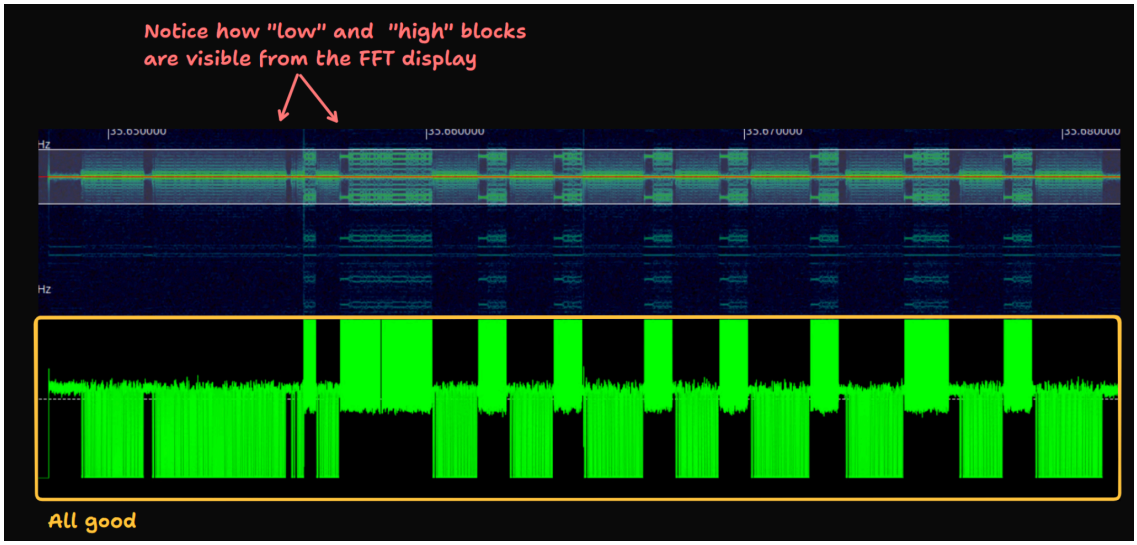
Ensure the center of the range is centered on the center of the transmission.



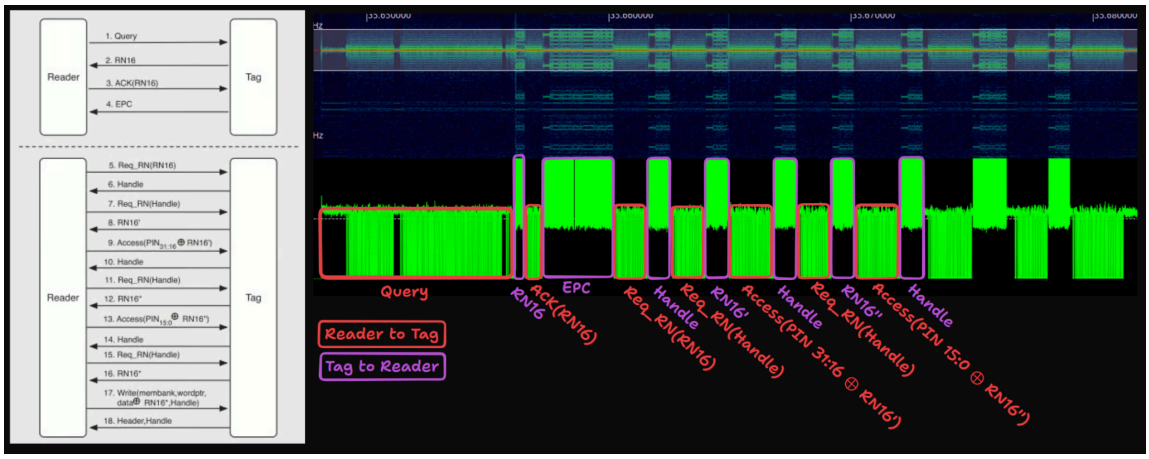
Lower the **power max** until the amplitude graph shows blocks. You should see blocks throughout the entire transmission and there should be “low” (reader) and “high” (tag) blocks.



The “low” and “high” blocks are also visible from the FFT display making it easier to find quality transmissions.



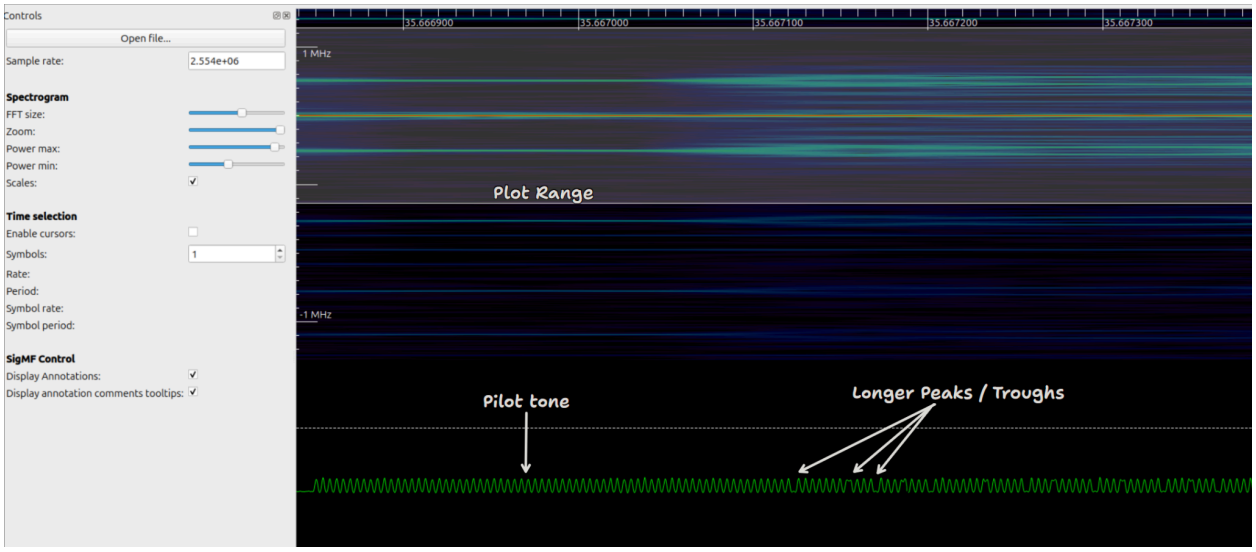
Using the chart in “Practical Eavesdropping” we can block out each packet in the transmission. The packets we care about are **RN16'**, **Access(PIN 31:16 ⊕ RN16')**, **RN16''**, and **Access(PIN 15:0 ⊕ RN16'')**



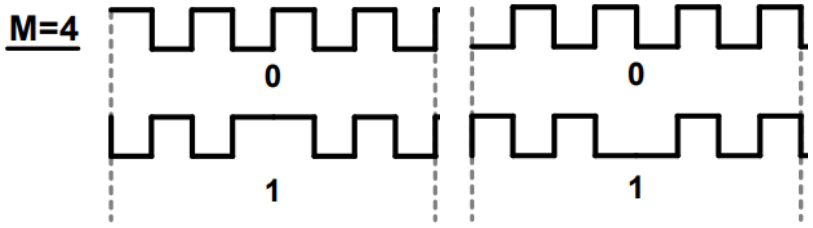
The password needed to write to the tags has its 16 most significant bits sent right after the **RN16'** packet and is encrypted by **PIN 31:16** ⊕ **RN16'** which is a bitwise XOR on the two numbers. The first step should be to record and decode the tag's response to the **Req_RN** commands which should provide us with both **RN16'** and **RN16''**. Then we should record and decode the reader's **Access** commands from which we can extract the encrypted **PIN 31:16** ⊕ **RN16'** and **PIN 15:0** ⊕ **RN16''**. We can then get the original password by performing a bitwise XOR over the encrypted password. This is because the bitwise XOR operator "cancels out" if applied twice.

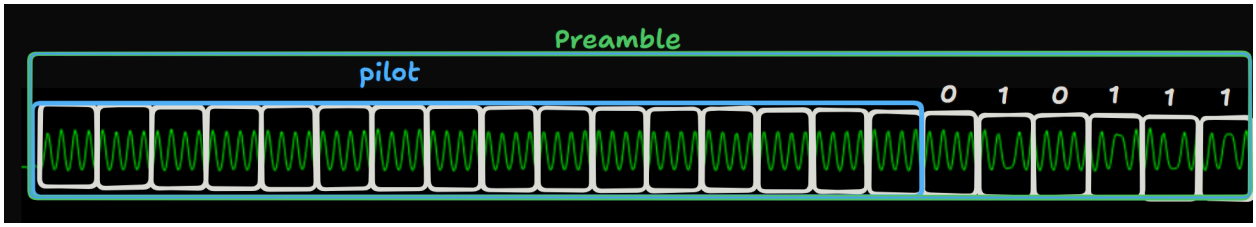
Analyzing the Tag's Response

To decode an **RN16** packet, start by identifying your packet and setting the **zoom** slider to max. Adjust the **power max** slider and the **plot range** until you can see an alternating signal. The signal should start with symmetric oscillations and then after a while you should see longer peaks or troughs.

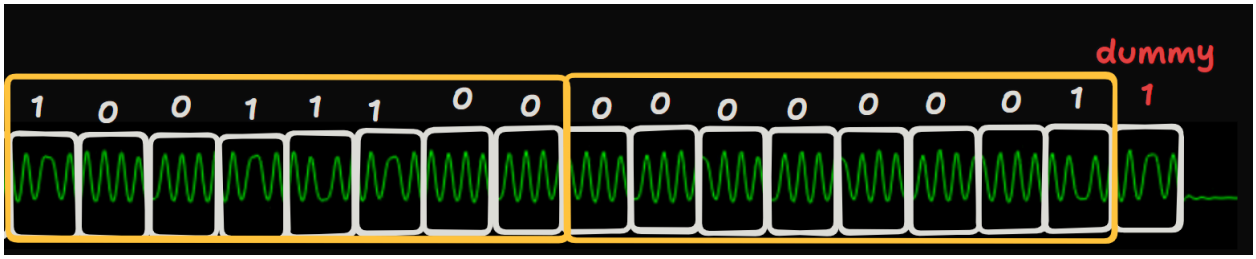


Using the information in the official EPC standard, **section 6.3.1.3**, we can determine that the tag is using Miller-modulated subcarrier with M = 4 and a RText = 1 preamble. "0"s are encoded by 4 equal high or low pulses while "1"s are encoded by 3 high or low pulses, the center pulse being twice as long.





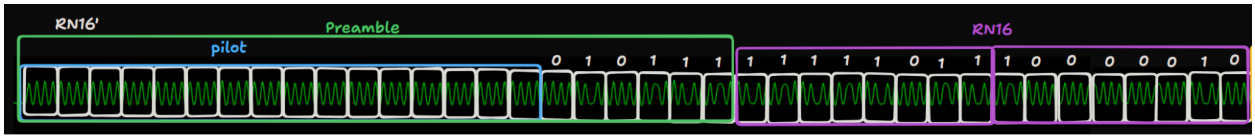
The rest of the signal can be blocked out in a similar way using the sequence chart as a guide. The Miller protocol also calls for a dummy “1” bit at the end of the packet.



Because the packet we just decoded is an **RN16**, the previous command should have been **Req_RN**. We can look this up in **section 6.3.2.11.3** to find the data structure of the packet. We see that the tag’s response defines the first 16 bits of the message as the **RN16**. These bits are important and should be set aside for now.

Table 6.31 – Tag reply to a *Req_RN* command

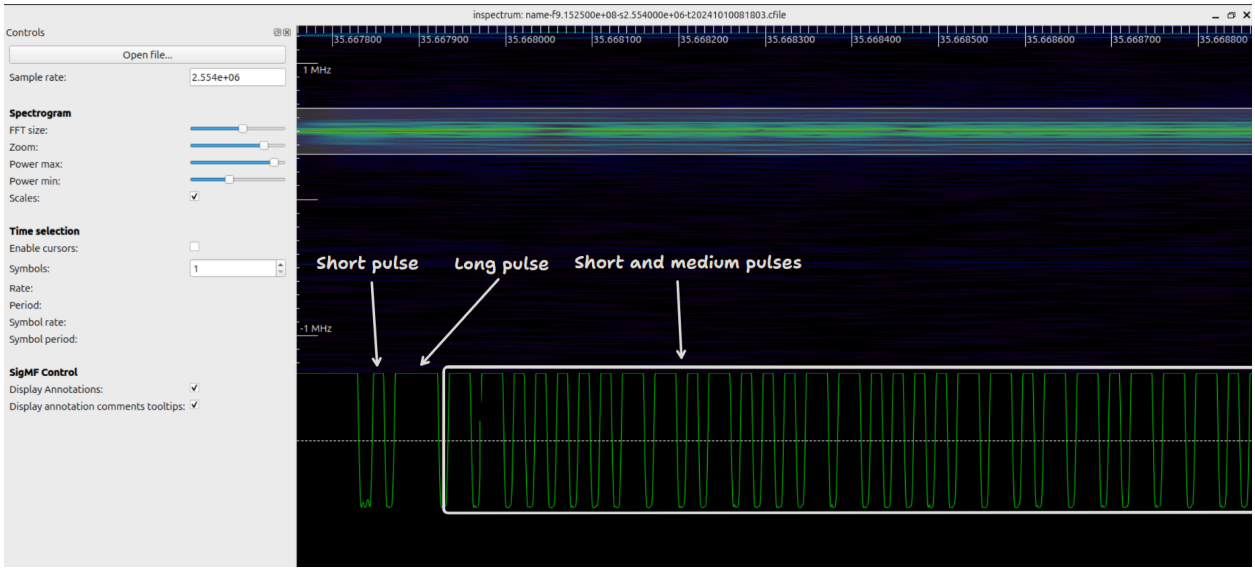
	RN	CRC-16
# of bits	16	16
description	handle or new RN16	



This process should be done once for **RN16'** and should be repeated for **RN16"**.

Analyzing the Reader's Response

To decode an **Access** packet, start by identifying your packet and setting the **zoom** slider to max or a few ticks before max. Adjust the **power max** slider and the **plot range** until you can see an alternating signal. The signal should start with one short pulse, one long pulse, and then continue with a combination of short and medium pulses.



Using the information in the official EPC standard, **section 6.3.1.2**, we can determine that the reader is using Pulse Interval Encoding (PIE) with an R=>T Frame Sync rather than a preamble.

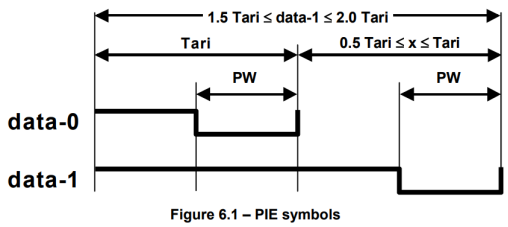
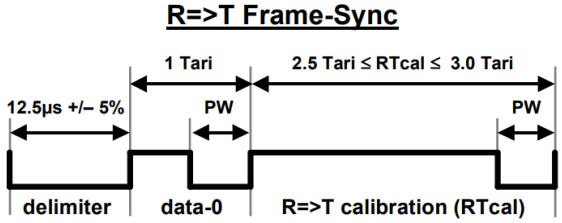
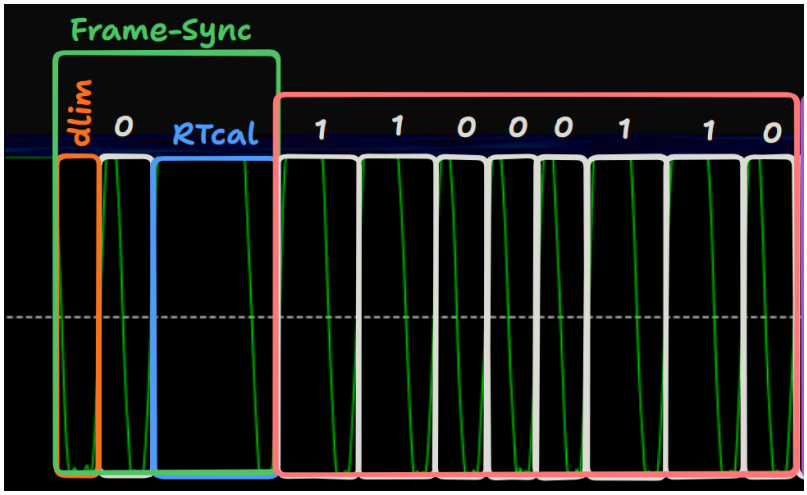
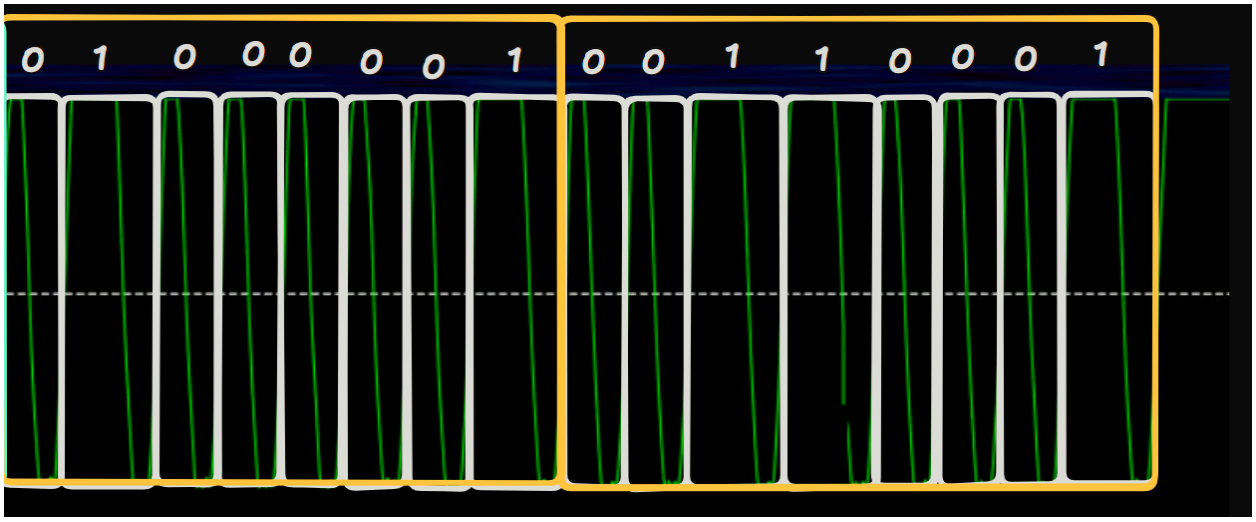


Figure 6.1 – PIE symbols





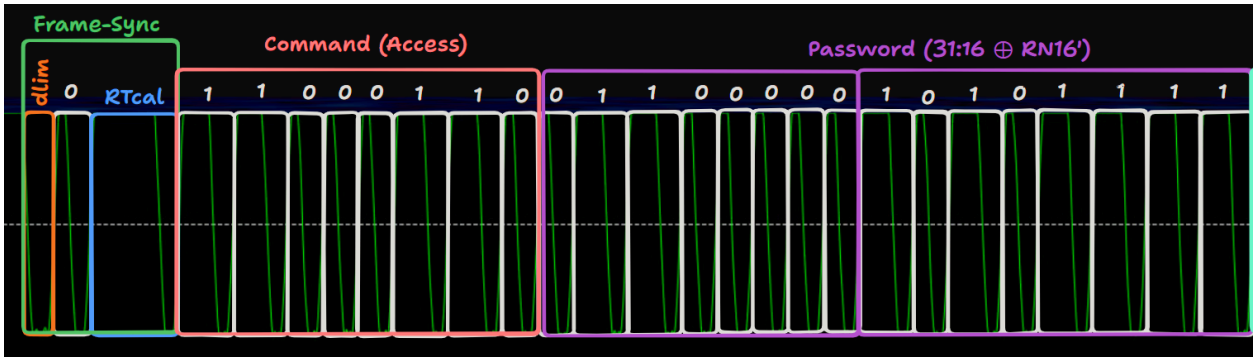
The rest of the signal can be blocked out in a similar way using the symbol chart as a guide. Short pulses are “0”s and long pulses are “1”s. The PIE protocol does not end with a dummy bit.



Because the packet we just decoded is an **Access** packet, we can look this up in **section 6.3.2.11.3** to find its data structure. We see that the **PIN** ⊕ **RN16** is defined by the 16 bits after the command. These bits are important and should be set aside for the next step.

Table 6.44 – *Access* command

	Command	Password	RN	CRC-16
# of bits	8	16	16	16
description	11000110	(½ access password) ⊗ RN16	handle	



This process should be done once for **PIN 31:16** ⊕ **RN16'** and should be repeated for **PIN 15:0** ⊕ **RN16"**.

Decoding the Password

To decode **PIN 31:16** from **PIN 31:16** ⊕ **RN16'** and **RN16'**, we'll have to perform a bitwise xor on the two binary numbers.

In our first analysis, we found that **RN16'** is "11111011 : 10000010"

In our second analysis, we found that **PIN 31:16** ⊕ **RN16'** is "01100000 : 10101111"

To perform a bitwise xor, for each bit in both numbers, we will compare the two. If they are the same (both "1"s or "0"s) the resulting bit will be "0". If they are the different (one is "1" and the other "0") the resulting bit will be "1".

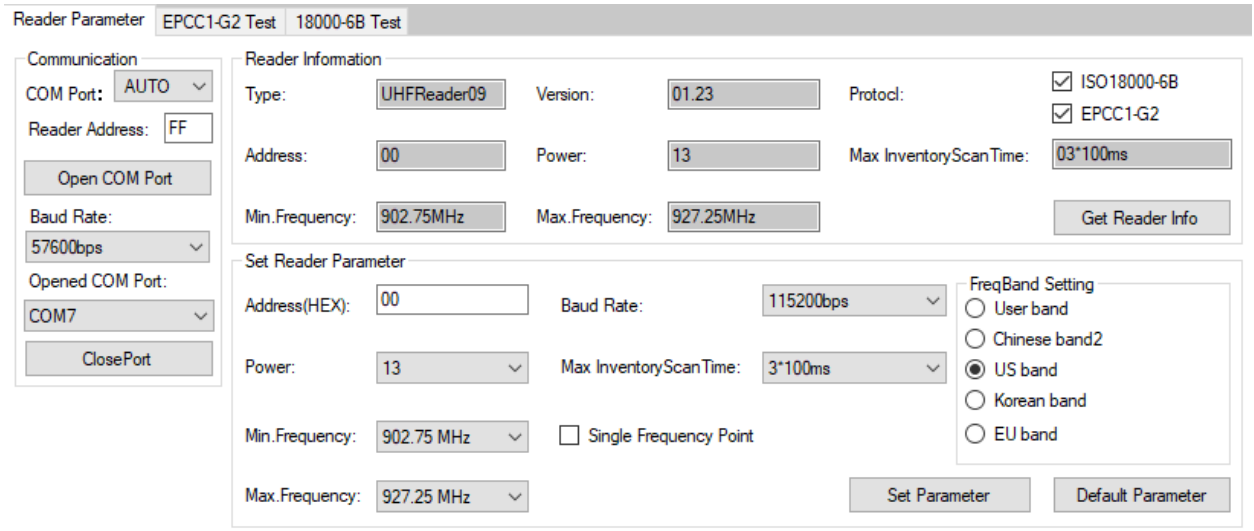
In this case our resulting **PIN 31:16** is "10011011 : 00101101"

We'll pretend that we calculated **PIN 15:0** as "11011010 : 00010011" which means our example password is 10011011001011011101101000010011.

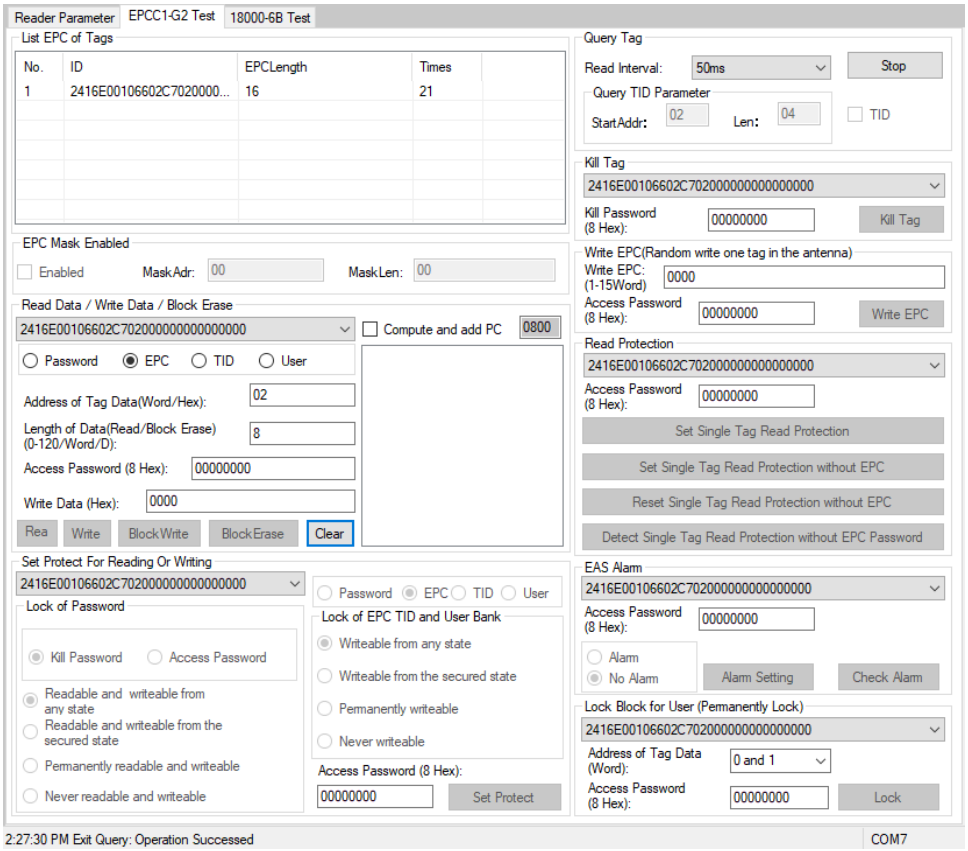
To use this with the reader's program, we'll need to convert it to hexadecimal which turns out to be **9B2DDA13**.

Reading and Writing to the Tag

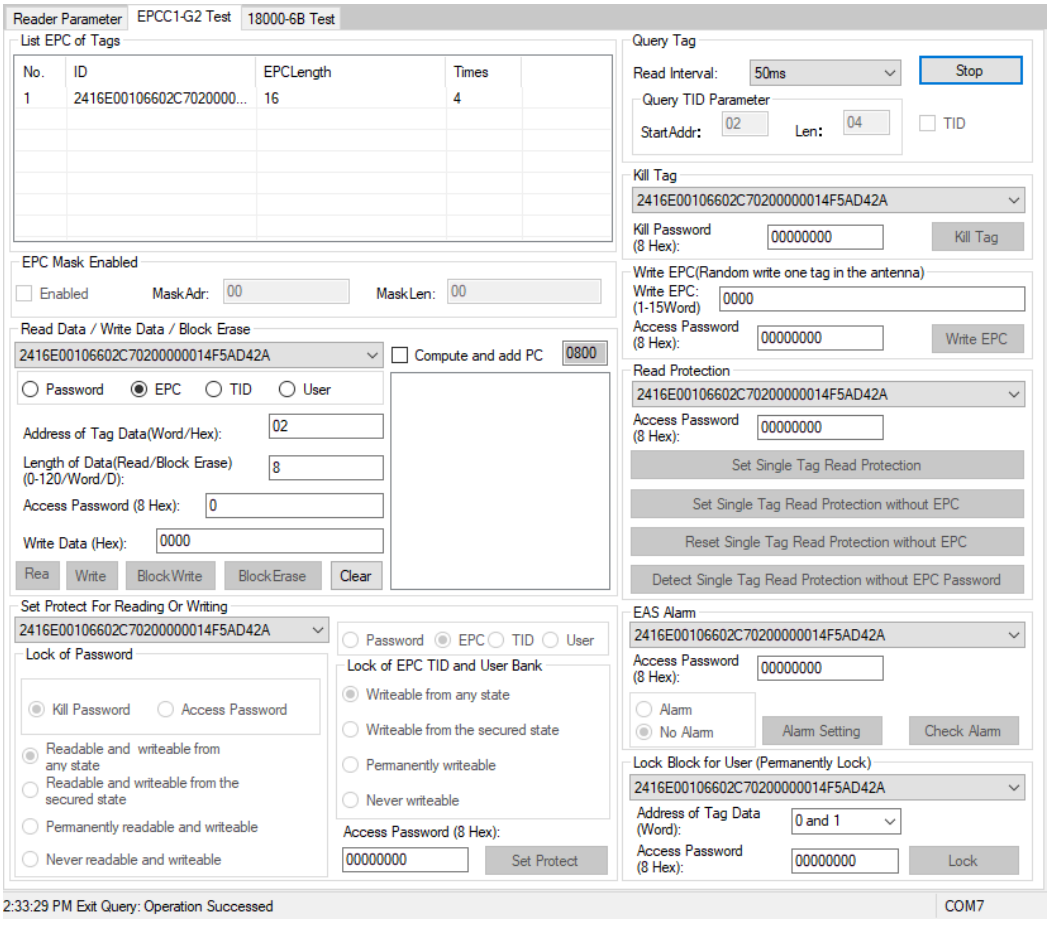
Start by opening the **PISwords demo program** and clicking open port. It may take a few minutes to establish a connection.



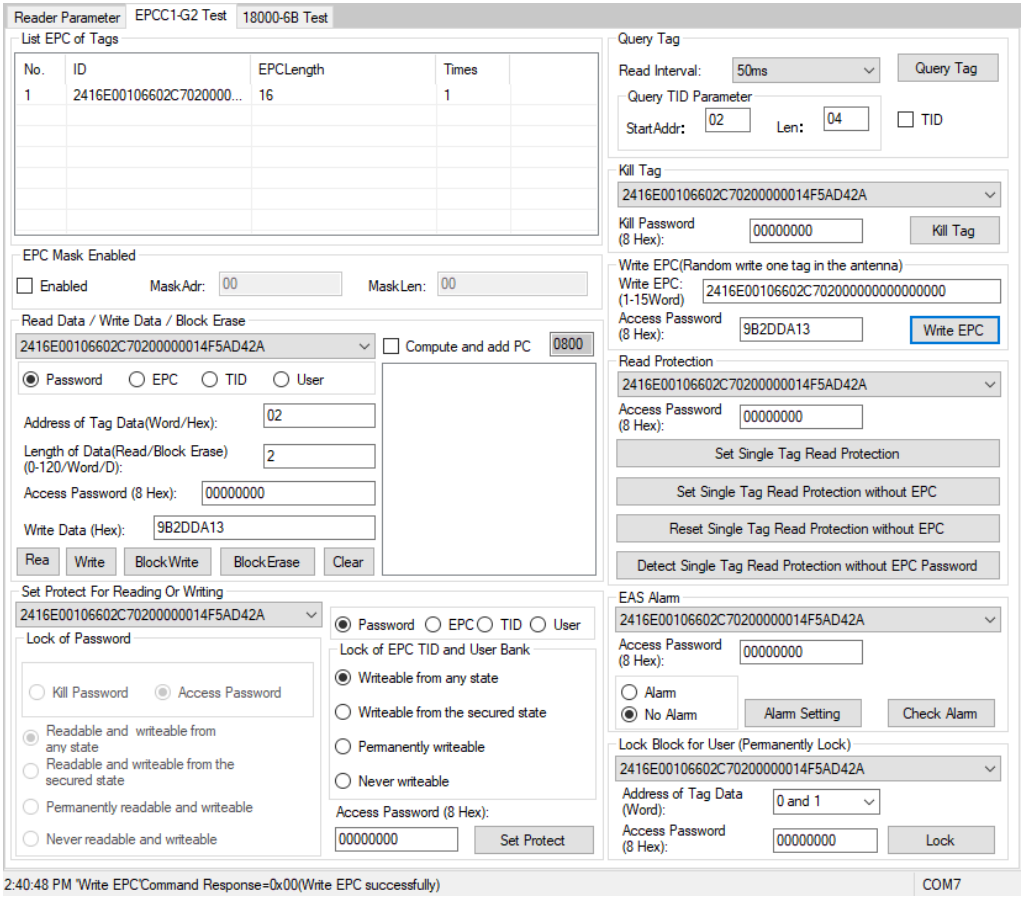
Switch to the **EPCC1-G2 Test** tab and grab a **new** tag. Click **query tag** in the upper right corner of the program. Bring the tag near until the reader beeps, you should see the tag's EPC being written on the list. This string encodes the number of fills on the bottle and should be noted for the next step.



Click **query tag** again and bring the tag that you wish to write to near the reader until it beeps. You should see the tag's EPC being written on the list. This step sets the target of the reader to the old tag.



Under the **Write EPC** section on the middle right, paste the EPC from the **new tag** under the **Write EPC** field and paste the password we discovered under the **Access Password** field. While the tag is in range, press the **Write EPC** button. You should hear a beep and message at the bottom of the program saying “Write EPC Successfully”



The tag will now act identically to the new tag that had been copied from.