**Wireless Power Transfer: Side-Channel Attacks and Mitigations**

Quinn Henry, Cole Hermanson, Nathan Pak, Alexander Singh

**Introduction**

Wireless power transfer is a relatively new technology that is being implemented in most handheld devices today. Not only is wireless charging considered a premium feature on most devices, but it is also critical for larger-scale technologies such as pacemakers and electronic cars. With its novelty, however, comes a lack of understanding of how it works and potential vulnerabilities that can be exploited. This capstone project has focused on discovering these exploitations to be able to mitigate them and create a safer technology in the long run. Currently, there are two feasible exploitations for wireless power transfer: hijacking and eavesdropping. Hijacking involves crafting a malicious message in a form that matches the Qi protocol the power transmitter (charger) and receiver (phone, headphones, etc.) use to communicate with each other. Doing this allows a user to stop the charging of the device, overcharge the device, or accomplish denial of service. Eavesdropping is like wiretapping and allows critical information about the device to be discovered. This attack vector can give a user charge status of the phone, a unique serial number, the manufacturer of the device, and all the power the phone has received over time. This combination of information can be used to predict what the device is doing. Overall, this project has deemed a potential risk of using wireless power transfer technology and research will continue in following years.

**Signature Analysis**

The Qi wireless charging protocol has become the industry standard for wireless charging as it provides high speed power transfer between a transmitter and a receiver[[1]](https://www.zotero.org/google-docs/?gtMzKu). This transmission is as fast as it is because the protocol lacks a method of securing the messages sent. Because of this, sensitive information can be read from the communication given the right tools. This information can be analyzed to make larger scale predictions based on signatures a device creates.

For the analysis of Qi signatures, six pieces of technology were required. A Yootech Wireless Charging Pad Model F500 was used as the wireless power transmitter, a Samsung Galaxy S7 was used as an Android receiver, an Apple iPhone 14 was used as an Apple receiver, an Avnet Engineering Services Qi Sniffer v1.3 was used to read all the communication sent between the transmitter and receiver, the Avent Engineering Services Qi Development Software was used to read and categorize the raw data of the communication, and Python was used to visualize and interpret the data. In combination, these technologies allowed for the generation and analysis of Qi message signatures.



Figure 1: Equipment setup

On the most basic level, there are three packets that are important for observing Qi signatures: identification packets, charge status packets, and power transfer packets[[2]](https://www.zotero.org/google-docs/?h8Sgvx). A comprehensive list of all the headers and all of the bytes is in the Wireless Power Consortium Qi WPT Power System Class 0 document. It outlines the hex designations for each Qi header to define what kind of packet it is and what it is doing. Identification packets can help uniquely identify a device on a wireless charger. The identification phase starts as soon as a device is placed on a charger and a connection is established. In this identification phase, the receiver and transmitter both provide an identification packet to ensure they are both compatible with each other. From experimentation with the Qi sniffer, identification packets from the receiver contain a header value of 0x71 and provide information about the version of qi the device is using, the manufacturer, and a unique serial code[[2]](https://www.zotero.org/google-docs/?T7ZVb2).

While most newer devices are updated to use the most current Qi version (1.3), the age of a device can be estimated if they are still on previous versions like 1.2 which was released in 2015 or 1.1 which was released in 2012. For example, the oldest device we tested was a Samsung S7 (released in 2016) which was observed to still be using version 1.2. Version 1.3 provides the most useful information as devices on the older versions do not send a charge status packet which states the exact percentage of battery the device has. A new version of Qi, Qi2, is scheduled to release in 2023. As of now, there is no information about new security features that will be included with the new release. The update focuses on creating a magnetic connection between the transmitter and the receiver to increase the efficiency of charge. (<https://www.businesswire.com/news/home/20230103005082/en/New-Qi2-Standard-for-Wireless-Devices-Ensures-Enhanced-Consumer-Convenience-and-Efficiency>)

As a device charges, many power packets are sent between the transmitter and the receiver. During this communication, the receiver sends control error packets to the transmitter to tell it how much power it needs for the task it is currently accomplishing. By plotting these values over time, predictions can be made about the device. For example, when a device is closer to full battery, less power is requested by the device. This is characterized by a large amount of negative control error packets indicating that the transmitter should reduce the amount of power it is sending. Additionally, as our experimentation shows, as the power decreases, the control error packets also request changes by a smaller amount, creating a smoother slope when wattage is plotted against time.

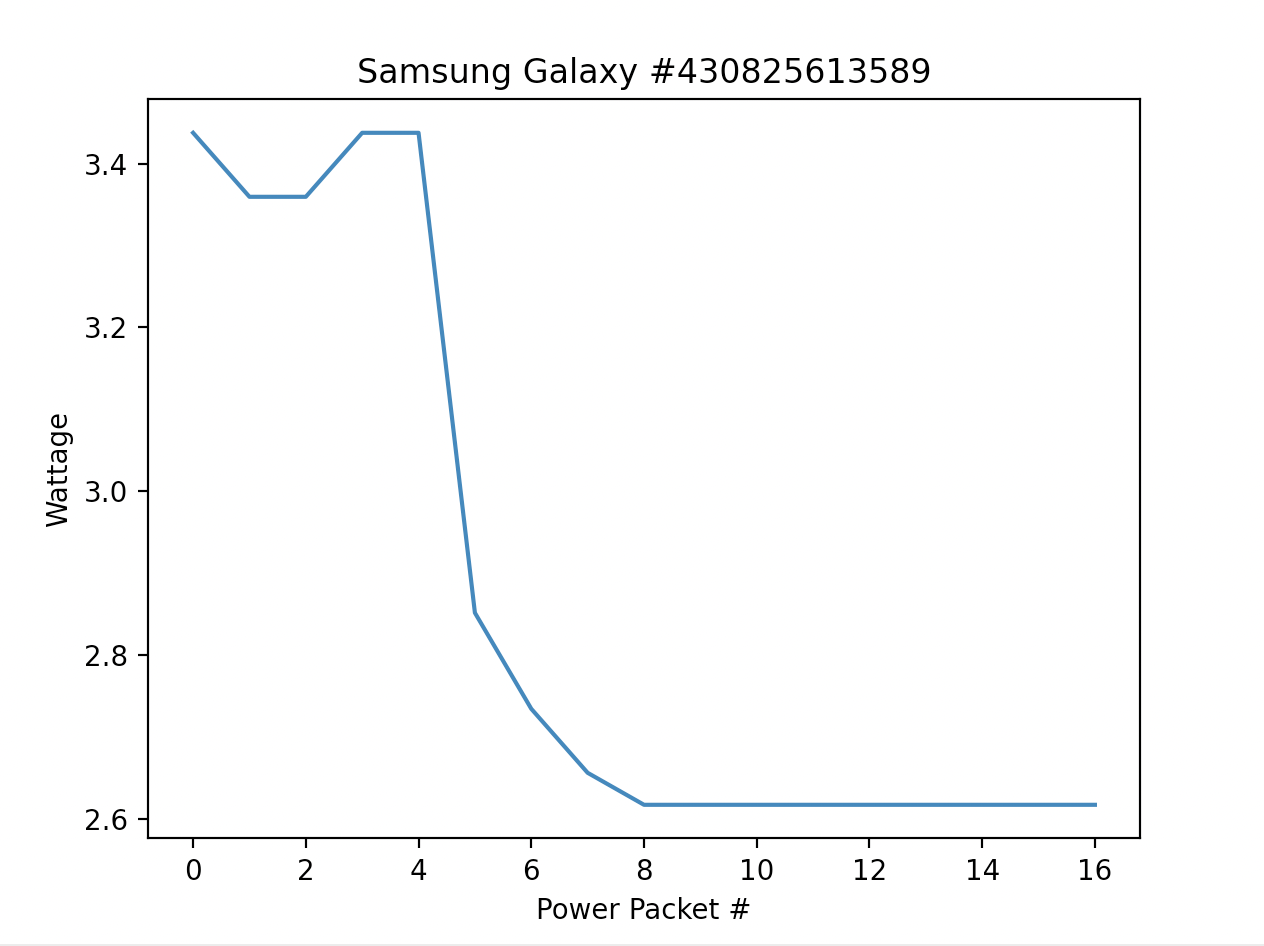
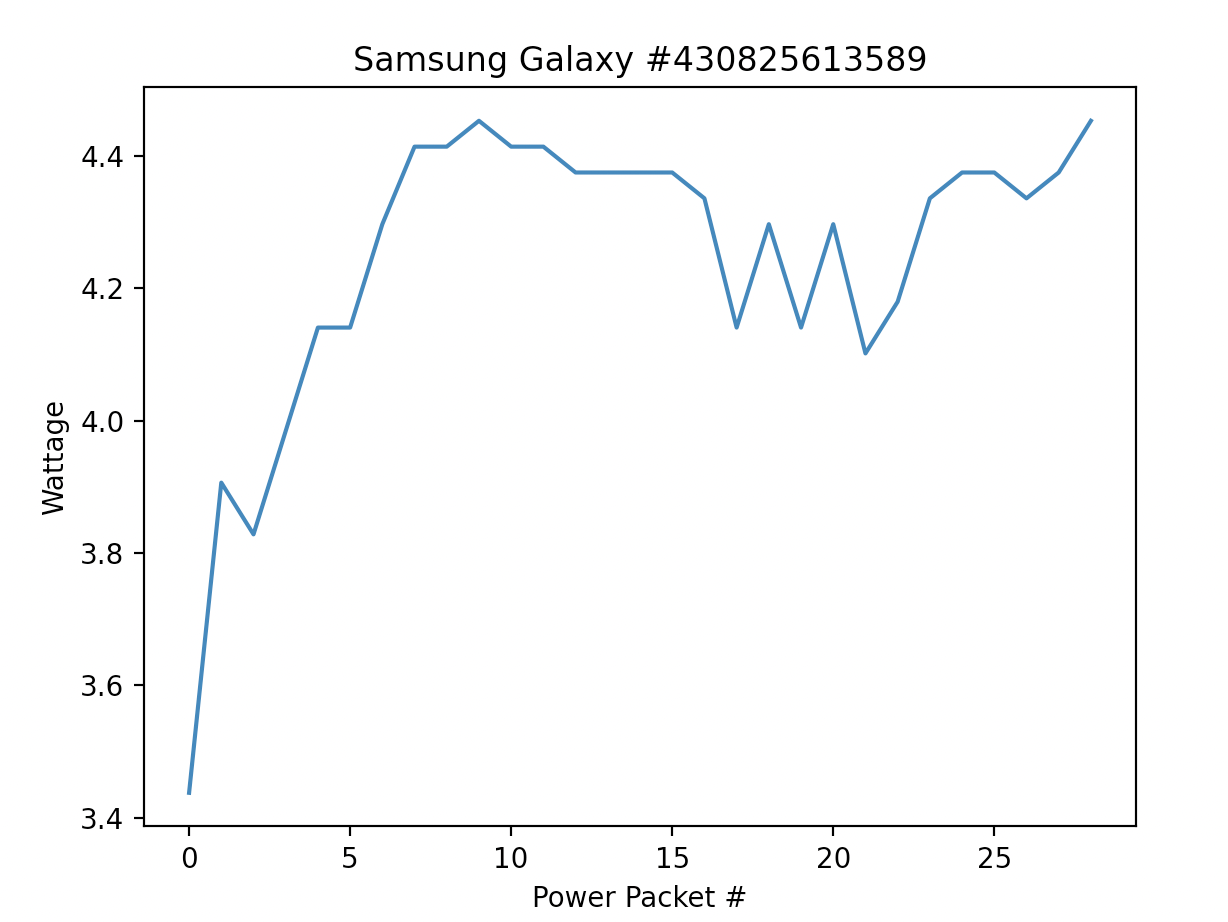


Figure 2: YouTube on low power (left) vs YouTube on high power (right)

During our experimentation, we tested the device while it was performing four functions: idle with screen off, idle with screen on, text messaging app, and YouTube. These applications were also recorded at three different power levels: full battery, medium battery, and low battery. The full battery experiments proved that when the phone reaches 100% battery, the transmitter quickly changed to a lower wattage output to account for a lower level of power needed by the phone. However, when YouTube videos were being played, a higher level of power was maintained as opposed to when the phone was on idle with the screen off.

Graphical user interface, application, table, Excel

Description automatically generated

Figure 3: Identification Packet Findings

**Encryption**

The Qi wireless charging protocol has become the industry standard for wireless charging as it provides high speed power transfer between a transmitter and a receiver[[1]](https://www.zotero.org/google-docs/?dVchUl). Due to the perceived lack of important information contained in the communication during the charging process, the communication during a Qi charging process is left unencrypted and open air. Due to this shortcoming, with the proper technique and devices, this communication can be intercepted and interpreted in order to conduct side-channel style attacks on the process. Small-scale encryption schemes exist specifically to address such a problem. A small-scale encryption scheme could then be used to encrypt Qi traffic and add a layer of protection to the process of WPT (Wireless Power Transfer).

For the design and implementation of a small-scale, lightweight encryption system using a Linear-Feedback Shift Register[[3]](https://www.zotero.org/google-docs/?FHss6L), we used the following software for a proof-of-concept demonstration: Matlab (v2023a), Simulink (v10.7). Simulink is a simulation software that runs via the program Matlab to create a block-diagram environment in which designs may be created and tested. For proof of concept, Simulink was used to recreate a circuit that implements a Linear-Feedback Shift Register encryption scheme.

To create a proof of concept for an encryption scheme to harden the Qi protocol, a decision had to be made as to the complexity of the encryption scheme in order to ensure that the timing of the Qi standard was not completely changed by the introduction of this new component. To this end, Bluetooth and its short-term key (STK) generation was evaluated against a Linear-Feedback Shift Register style encryption. It was evaluated that while STK provides a safer option mathematically, it would be more invasive to incorporate into the standard and could be potentially slower in operation than LFSR. This is due mainly to the requirement by STK for each device to perform a Diffie-Hellmann key exchange to begin the encryption process, then to independently compute the short term key and compare this key with the other device to ensure the key is in fact the same[[4]](https://www.zotero.org/google-docs/?34xZ9h). LFSR does not require this increased computational burden on the devices and as such can be definitively considered a more lightweight process. For this reason LFSR became our choice for an encryption scheme in order to harden the Qi standard from snooping style attacks.

The Linear-Feedback Shift Register (LFSR) method for encryption takes advantage of a constant stream of pseudo-randomly generated bits, known as the stream cipher key, with which the message may be XOR’d to produce an encrypted result. This encrypted result can then be sent to another device utilizing the same stream cipher key, meaning that both devices will need to compute the stream cipher key independently and at the same time. By utilizing XOR on the encrypted message and the stream cipher key, the message may be decrypted. An LFSR stream cipher is not the most mathematically sound encryption method and has many vulnerabilities that can lead to cracked communications, however for the purposes of simply adding to the complexity of intercepting Qi communications and providing a solid proof of concept for Qi encryption, an LFSR stream cipher works very nicely.

The creation of the circuit in Simulink is fairly easy to accomplish and simply requires: a generated stream cipher, a message stream of bits, an XOR gate taking the message and the stream cipher as inputs (this would then output the encrypted message), and another XOR gate taking the encrypted message and the stream cipher as inputs (which would then output the decrypted message). In figure 1, a working example of this circuit is shown (with all outputs linked to a display for easier UI).

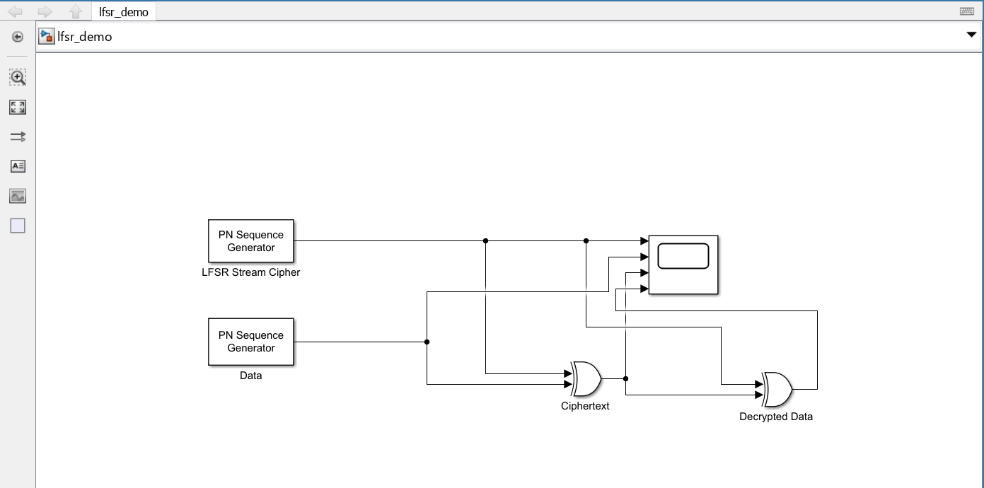


Figure 4: Depicting an example of an LFSR Stream Cipher in Simulink

If the above circuit is run, the following result is produced (see figure 2).

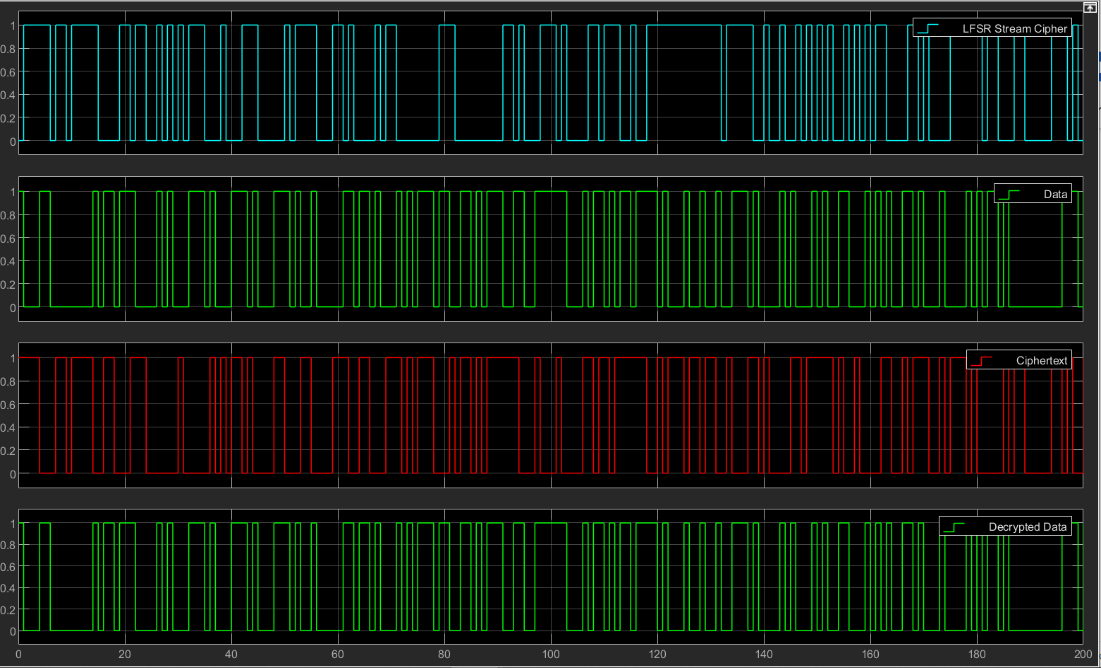


Figure 5: Depicting the outputs of the LFSR Stream Cipher Simulation

The results produced by the Simulink simulation show that the data (the second row, labeled “Data” in Figure 2) used as the message is successfully encrypted (shown in the third row labeled “Ciphertext” in Figure 2). Then by juxtaposing the decrypted data (last row labeled Decrypted Data in Figure 2) with the original data or message (second row in Figure 2), it can be determined that the ciphertext was successfully decrypted by the XOR gate and as such the encryption scheme works.

This may be an easy concept to replicate in Simulink, but when dealing with multiple devices not connected physically as the Simulink circuit depicts the problem becomes more complicated. A clock (typically a function of hardware located inherently on both devices) will need to be synchronized between the two devices. After this synchronization process is complete, the stream cipher will have to be the same between the two devices (i.e. creating a shared key between them), this stream cipher, however, can be calculated by each device separately. Once the stream cipher begins to cycle (iterating through bits on the clock cycle), they can be used as an input against a message in an XOR function.

A few methods exist to further complicate this process to avoid an easy negation of the encryption efforts. One could, for example, use an irregular clock which would ensure that the clock also must be replicated and not simply the stream cipher itself to break the encryption scheme [[5]](https://www.zotero.org/google-docs/?3q3rE0). Another method is using a more complicated stream cipher algorithm, this will result in less cycles of a stream cipher increasing the time it would take to brute force list the bits in a stream cipher. The most effective stream cipher to use for this purpose is widely regarded as the Fibonacci stream due to its exceptional stream length and easy calculation [[5]](https://www.zotero.org/google-docs/?5gPnv0).

From the Simulink proof of concept simulation, it can be inferred that this type of encryption is not only quick enough to be integrated into a Qi system, but also secure enough to stop open air interception without key and clock interception as well. This would increase the burden on potential snooping efforts and potentially limit the risk involved with using Qi standards.

**Alternative Encryption Methods**

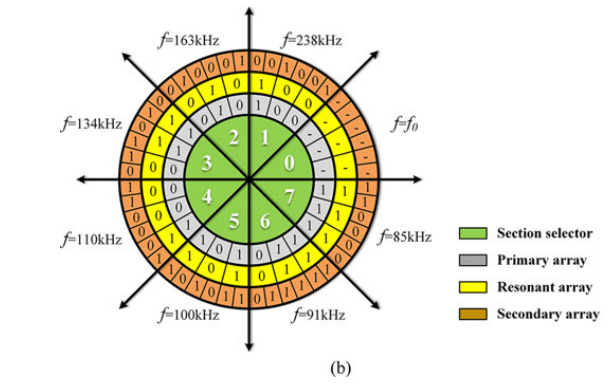
In addition to LFSR, another possible encryption method takes advantage of an energy transmission scheme enabled by magnetic resonant coupling (MRC) [[6]](https://www.zotero.org/google-docs/?eRriKm). Due to the difference in WPT schemes, this MRC-based encryption method would not suit the Qi protocol, as it uses induction. Efficient energy transmission relies heavily on the switching frequency of the power transmitter. When the frequency deviates from the optimal switching frequency, transferred power can be suppressed, causing charging to be inefficient [[7]](https://www.zotero.org/google-docs/?al9P2L). This encryption method utilizes this sensitivity in a positive way by adjusting the optimal switching frequency and communicating the adjustments to the receptor. The frequency is adjusted to a predefined regulation and is kept confidential and unpredictable to all other power receptors.

This method is also known as chaotic frequency regulation as the frequency is regulated unpredictably [[7]](https://www.zotero.org/google-docs/?wqNeSs). The chaotic sequence acts as a key to encrypt the transferred energy. The switching frequency is chaotically varied within a range, which is arbitrarily chosen accordingly with the range and transmission distance [[7]](https://www.zotero.org/google-docs/?koqgpO).

In an MRC-based WPT scheme, there are three capacitors to be utilized: primary, resonant, and secondary. Each capacitor has a capacitor array, where the primary capacitor has 3 parameters, resonant 3 parameters, and secondary 6 parameters [[7]](https://www.zotero.org/google-docs/?y4J3ca). The encryption method works in eight different resonant coupling states using a logistic map given by the following equation:

εi = floor(8ξi)

Where ξi is the chaotic logistic sequence. Figure 3 shows a predefined switching table, the on-off states of the capacitor arrays of its respective capacitor, and the switching frequency that the power transmitter shall regulate to [[7]](https://www.zotero.org/google-docs/?9s0OL0).



Signal Injection:

Figure 6: Switching table diagram [[7]](https://www.zotero.org/google-docs/?5A1tW3)

While this proposed encryption scheme fulfills encryption requirements and would secure communications between the power transmitter and receiver, it would not be appropriate for the Qi protocol which uses induction rather than magnetic resonant coupling.

**SIgnal Injection**

Injection attacks against Qi-enabled devices include the existing transmission and receiver communication and associated power transfer with an adversary-controlled signal introduced and overlayed the existing communication that attempts to either outright disrupt communications or insert manipulated/new data into the transmission [[8]](https://www.zotero.org/google-docs/?XPJjSj).

For the purpose of our experimentation, signal injection was first tested and demonstrated via a proof-of-concept application drawn from “Wireless Charging Power Side-Channel Attacks” [[9]](https://www.zotero.org/google-docs/?WWCmks). Signal injection was facilitated by the Keysight 33500B Series Waveform Generator (Depicted in Appendix A). The waveform generator passes the crafted waveform via an adversarial coil. The proof of concept sought to demonstrate the application of injection by overlaying a 0-1-0-1 repeated binary signal on top of an existing waveform. This pattern was chosen because it was an easily recognizable pattern that was distinctly represented in the captured waveform and something that would indicate the attempted injection was successful. This was captured via the oscilloscope with the captured waveform depicted in Figure 4.



Figure 7: Signal Injection Proof of Concept

With the completed proof of concept the next step was crafting customized signals to interact with existing transmissions. Utilizing Python, we created a script (included in Appendix B) to craft custom signals. These included an End Power Transfer (EPT) and Control Error (CE) Control 1 packet. Each packet is divided into four parts: a preamble consisting of 11-25 bits, a header which denotes the type of packet being sent, the data message, and the packet checksum. These crafted signals are stored in a .csv file format which can be loaded onto and interpreted by the waveform generator. Once the crafter signal is passed through the waveform generator to the adversary coil, the injected signal interaction was captured using the Qi sniffer. Figure 5 shows a captured interaction with an injected signal.

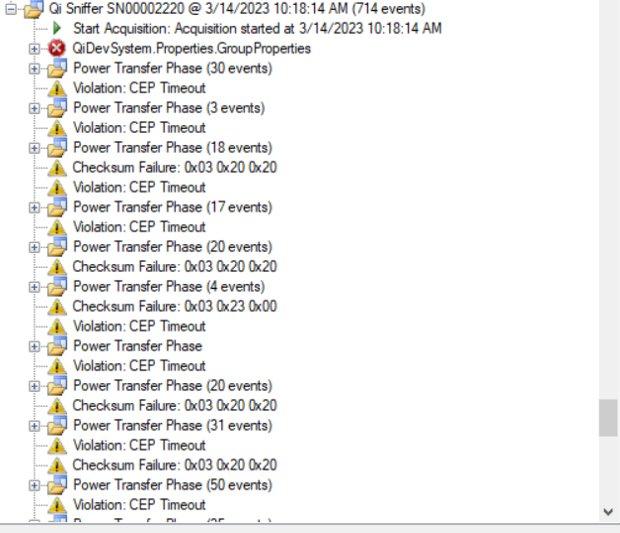


Figure 8: Injected Signal Capture

Of note in the above capture is the continued checksum failure on the attempted Control Error Packet injection. This indicates the overlapping of the injection signal with the existing signal being transmitted. A challenge in injecting signals is the fact that crafted packets are not sent individually but alongside the current communication between the transmitter and receiver. This means one must account for how the injected signal will overlap with the existing signal and ensure any packet modifications are properly represented in a newly calculated checksum value. This is an assumption based upon existing experiment observations and will need to be tested and confirmed in future research. Our current research began exploring how injection works but was unsuccessful in properly overlaying injection signals. This would be an important area for future research.

The ultimate goal of signal injection ranges from attacks such as denial of charge and device over charging to more advanced possibilities such as writing data to a victim system or querying specific information.

**Conclusion**

From our experimentation, it can be concluded that the Qi wireless power transfer protocol as it exists right now is a vulnerable technology. It is able to be exploited via snooping and hijacking attack vectors which can cause information leak and denial of service. Our future work aims to expand these attack vectors to other similar technologies such as bluetooth, LiFi, RFID, optics, and lasers.

References

[[1] Wireless Power Consortium, “Introduction to the Power Class 0 Specification Version 1.2.3.” Wireless Power Consortium, Feb. 2017.](https://www.zotero.org/google-docs/?pMJHNq)

[[2] Wireless Power Consortium, “The Qi Wireless Power Transfer System Power Class 0 Specification Version 1.2.3.” Wireless Power Consortium, Feb. 2017.](https://www.zotero.org/google-docs/?pMJHNq)

[[3] A. Canteaut, “Lecture Notes on Error-Correcting Codes and their Applications to Symmetric Cryptography”.](https://www.zotero.org/google-docs/?pMJHNq)

[[4] A. Duque, “Deep Dive into Bluetooth LE Security,” Mar. 2018, [Online]. Available: https://medium.com/rtone-iot-security/deep-dive-into-bluetooth-le-security-d2301d640bfc](https://www.zotero.org/google-docs/?pMJHNq)

[[5] T. Suresh and M. Ramakrishnan, “Design of Low Lower NFSR for RFID System With Irregular Clock Pulse,” *Microprocessors and Microsystems*, vol. 73, p. 102983, Mar. 2020, doi: 10.1016/j.micpro.2019.102983.](https://www.zotero.org/google-docs/?pMJHNq)

[[6] X. Mou, D. T. Gladwin, R. Zhao, and H. Sun, “Survey on Magnetic Resonant Coupling Wireless Power Transfer Technology for Electric Vehicle Charging,” *IET Power Electronics*, vol. 12, no. 12, pp. 3005–3020, 2019.](https://www.zotero.org/google-docs/?pMJHNq)

[[7] Z. Zhang, K. Chau, C. Qiu, and C. Liu, “Energy Encryption for Wireless Power Transfer,” *IEEE Transactions on Power Electronics*, vol. 30, no. 9, pp. 5237–5246, 2014.](https://www.zotero.org/google-docs/?pMJHNq)

[[8] Y. Wu, Z. Li, N. Van Nostrand, and J. Liu, “Time to Rethink the Design of Qi Standard? Security and Privacy Vulnerability Analysis of Qi Wireless Charging,” presented at the Annual Computer Security Applications Conference, 2021, pp. 916–929.](https://www.zotero.org/google-docs/?pMJHNq)

[[9] A. S. La Cour, K. K. Afridi, and G. E. Suh, “Wireless Charging Power Side-Channel Attacks,” presented at the Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, 2021, pp. 651–665.](https://www.zotero.org/google-docs/?pMJHNq)

Appendix A: Keysight 33500B Series Waveform Generator with adversarial coil



Appendix B: waveform\_generator.py

|  |
| --- |
| # Create a custom qi packet wavefrom that can be loaded onto a waveform generator and used as an injection payload for qi # charging disruption attacks  import numpy as np import matplotlib.pyplot as plt import binascii  PREAMBLE\_BIT\_SIZE = 16 CLOCK\_FREQUENCY = 150 SAMPLING\_RATE = 44100  def generate\_packet():  # craft a power control packet that uses the qi protocol   # packet preamble (11-25 bits all set to 1)  temp = 0xfffffff  preamble = [1] \* PREAMBLE\_BIT\_SIZE # x bit preamble  print(f'Preamble: {preamble}')   # generate header  # EPT End Power Transfer Power control 1  ept\_packet\_header = 0x02  # CE Control Error Packet control 1  ce\_packet\_header = 0x03  header\_hex = ept\_packet\_header  header\_bin = bin(header\_hex)[2:]  print(f'Header: {header\_bin}')   # generate packet message  data\_message\_hex = 0x2357 # Example data message as hex  ce\_data\_message\_hex = 0x0000 # Example data message as hex  ce\_data\_message\_hex\_deplete = 0x049c96 #attempt to request -100 power  ept\_data\_message\_hex = 0x0000     selected\_data\_message\_hex = ce\_data\_message\_hex  data\_message\_bin = bin(selected\_data\_message\_hex)[2:]  data\_bin\_arr = [data\_message\_bin[i:i+8].zfill(8) for i in range(0, len(data\_message\_bin), 8)]  print(f'Data Message: {data\_bin\_arr}')    # generate packet checksum  # checksum is calculated by xoring the header and each byte of the data message  for byte in data\_bin\_arr:  checksum = int(header\_bin,2) ^ int(byte,2)  print(f'Checksum: {bin(checksum)[2:]}')   #assemble packet  packet = preamble + [int(x) for x in header\_bin] + [int(x) for x in data\_message\_bin] + [int(x) for x in bin(checksum)[2:]]  print(f'Packet: {packet}')   # Convert binary to list of integers  data = [int(x) for x in data\_message\_bin]  print(data)   # Existing signal data  data = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,]   #return packet  return data  def generate\_waveform():  # Define the clock frequency and sample rate  clock\_frequency = CLOCK\_FREQUENCY  sampling\_rate = SAMPLING\_RATE   data = generate\_packet()   # Define the time vector  t = np.linspace(0, len(data)/clock\_frequency, int(sampling\_rate/clock\_frequency\*len(data)), endpoint=False)   # Create the waveform  waveform = np.zeros(t.shape)  for i in range(len(data)):  if data[i] == 1:  waveform[int(i\*sampling\_rate/clock\_frequency):int((i+0.5)\*sampling\_rate/clock\_frequency)] = 15  waveform[int((i+0.5)\*sampling\_rate/clock\_frequency):int((i+1)\*sampling\_rate/clock\_frequency)] = 14  else:  if i % 2 == 0:  waveform[int(i\*sampling\_rate/clock\_frequency):int((i+1)\*sampling\_rate/clock\_frequency)] = 15  else:  waveform[int(i\*sampling\_rate/clock\_frequency):int((i+1)\*sampling\_rate/clock\_frequency)] = 14   # Export as csv  np.savetxt("crafted\_waveform.csv", waveform, delimiter=",")   return waveform, t  def graph\_waveform():  waveform, t = generate\_waveform()   # Plot the waveform  plt.plot(t, waveform)  plt.xlabel('Time (s)')  plt.ylabel('Amplitude (V)')  plt.ylim([0,20])  plt.title('Signal waveform')  plt.show()  def main():  #generate\_waveform()  graph\_waveform()  if \_\_name\_\_ == "\_\_main\_\_":  main() |