Project 5: Build a Sigfox application and launch a replay attack

AARJAV KOTHARI, [redacted], Univeristy of California Los Angeles
PRAGYA SHARMA, [redacted], Univeristy of California Los Angeles
MICHAEL SHERMAN, [redacted], Univeristy of California Los Angeles

Sigfox is an emerging Low Power Wide Area Network technology that serves long-range, power-constrained IoT devices. It is simple in its design and thus suitable for low-end IoT devices. It supports reliable, anti-jamming services as it employs diversity where each message is repeated for multiple times at different time and frequency. A Sigfox device does not need to establish connection with any base station; instead, any base station that captures the Sigfox data will process it. Therefore, it suffers from the packet replay attack: An attacker can capture a packet and replay it later (when certain condition is met). The receiver can falsely accept the replay from the attacker. In this project, we implement a Sigfox application and demonstrate a man-in-the-middle attack where an attacker node (USRP B210) captures and replays the Sigfox data.

Additional Key Words and Phrases: Low-power Wide Area Networks, Sigfox, GNU Radio, Man-in-the-middle

1 INTRODUCTION

Recent advances in city-scale wireless sensor networks have lead to the development of LP-WANs (low-power wide-area networks) and face connectivity as their main challenge. Traditionally, these wireless sensors are small, power-constrained devices with little processing capabilities. To enable a network of such devices to communicate among themselves, low-power and long-range protocols have emerged. Sigfox [5] is one such narrow-band wireless protocol that operates in the 900MHz range and uses Binary Phase Shift Keying (BPSK) i.e. it changes the phase of the signal to encode the data. As a result of this modulation scheme, the receiver only needs to listen to a small part of the spectrum mitigating the effects of wide band noise. Sigfox's simple design leverages signal diversity by replaying every transmission three times at a slightly shifted time and frequency. Drawing its inspiration from cellular communication, the client node does not need a lot of processing or information about the network to operate. All this computation is pushed to the Sigfox base station. However, unlike cellular technology, Sigfox protocol does not necessitate a handshake mechanism between the client and the server. While this reduces the computational and latency overheads, it opens the door to significant security vulnerabilities. Sigfox protocol uses two mechanisms to ensure secure communication between the client and the server nodes. First, only those clients that are provisioned with the network will be able to access it. The provisioning mechanism is fairly straightforward requiring the user to register the device ID on Sigfox server application. Second, as a part of packet-specific security, each payload is encrypted using 128-bit Advanced Encryption Standard (AES) mechanism. However, the protocol standard does not require authentication as a fixed header in the packet and the developer can choose to turn it off from both the client transmissions and the base station decoder.

Authors' addresses: Aarjav Kothari, 805526480, Univeristy of California Los Angeles, aarjavkothari23@ucla.edu; Pragya Sharma, 305033739, Univeristy of California Los Angeles, pragyasharma@ucla.edu; Michael Sherman, 605507996, Univeristy of California Los Angeles, msherman32@ucla.edu.

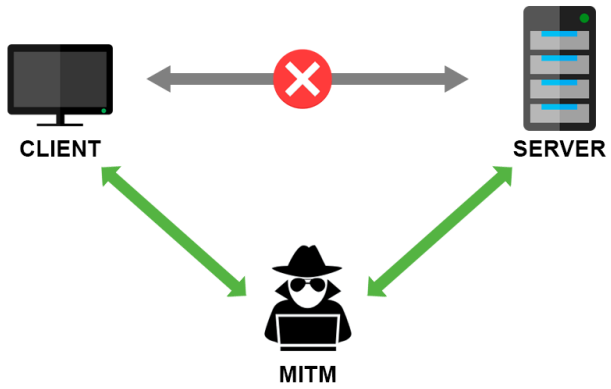

Fig. 1. Man-in-the-middle attack

These non-robust security measures leave room for various attacks on the network communication and system in general, especially in IoT applications which cannot afford to employ heavy security mechanisms. As shown in Figure 1, man-in-the-middle (MITM) attack is one of the most fundamental system security attacks where an attacker node snoops on the communication between the client and server. The attacker can then pretend to be the client node and get access to the network.
Our objective of this project is two-fold. First, we create a setup of two Sigfox-enabled devices and enable them to communicate with each other. A plug-and-play Pycom FiPy [3] acts as the client node while an SDR Dongle acts as the Sigfox Base Station. Our second objective is to demonstrate a MITM attack using USRP B210 [6] as the attacker node to record and replay the captured signal.
The rest of the paper is organized as below. Section 2 dives deeper into previous work in this area while Section 3 outlines the system design and components. We go over implementation challenges in Section 4 and briefly talk about project evaluation in Section 5. We finally conclude with the future direction of this work in Section 6.

2 BACKGROUND AND RELATED WORK

Sigfox is used heavily in industries where data is required to be transmitted over long distances but is not significant enough or, in some cases valuable enough, to be secure. One prominent example of Sigfox deployment is waste management systems. Ultrasonic sensors are used to monitor the waste levels in garbage bins and these sensor devices are connected using Sigfox. Another example is in dense areas, it can be challenging to find a parking place. Hence, more and more cities are adding Sigfox sensors connected to a smartphone application so that drivers can be routed to a vacant

parking place directly. We found it interesting that even Google has an IoT Cloud Github setup with Sigfox . There are tutorials that explain how to ingest data from Sigfox devices and publish it to the Google Cloud Services along with device configuration management and service message logging. The device configurations are stored in Firestore and logs in the Cloud Logging. This implementation is set to be low-cost, scalable and stateless[1].

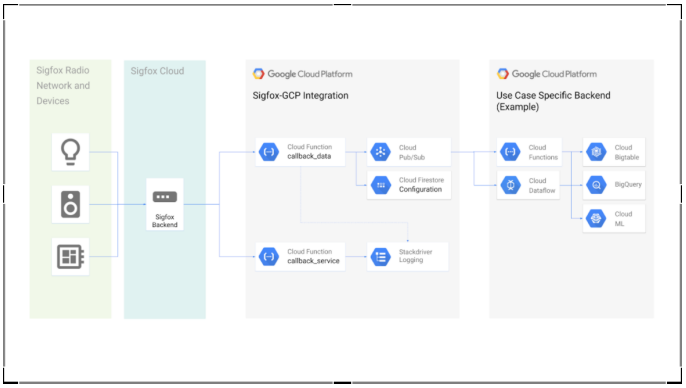

Fig. 2. Design diagram of integrating Sigfox with Google Cloud

Sigfox is privately owned by the parent company named the same and their documentation on how their network functions is not entirely open-source. While significant research is done on a high-level under this topic's umbrella, there are hardly any publications or reverse-engineered protocol solutions on the Sigfox network. This led us to wonder how this network functions but more importantly, how does packet-based authentication work over the network. This got us further questioning the methods by which we could attack the network and find any flaws if possible.

The existing research on Sigfox security attacks includes analysis as to how replay attacks could be performed against Sigfox base stations. For instance, according to the Coman et al., Sigfox uses its 12-bit sequence number protected by a Message Authentication Code (MAC) to prevent replay attacks from occurring since the Sigfox backend drops any packets with sequence numbers less than the highest, most-recently seen number. However, this does not imply that the Sigfox protocol cannot be abused to lead to a replay attack [7]. If an attacker were to capture and store all the packets intended for a Sigfox base station until the 12-bit sequence number reset from 4096 to 0, then the attacker could certainly be able to replay messages to the base station (even with daily uplink limitations of 140 messages in consideration). It would just take on the order of a month before the sequence number reset. It seems as though thorough research has been put into whether or not Sigfox suffers from replay attacks, but we decided to launch our own MITM attack for the purpose of this course project's objective.

3 IMPLEMENTATION GOALS

To achieve our project objectives, we design the system as seen in Figure 3

Fig. 3. System Design

The following subsections describe the three components we used - Sigfox Base Station (BS), FiPy client node, and the USRP attacker node.

3.1 Sigfox Test Bed

The Sigfox BS is a combination of the SDR Dongle hardware and the Sigfox Network Emulator (SNE) software solution. The SDR Dongle is a USB stick that connects directly to a computer and uses an external antenna. The SNE emulates a Sigfox network and is not connected to the Sigfox backend. We choose a lightweight FiPy board to act as the Sigfox client. In order to enable communication between the client and the server, we first retrieve the identity (device ID, in this case) of the FiPy and register it with the Sigfox SNE. Once this connection is established and the SNE is running, we can ping our Sigfox base station by running a communication script that we developed on the client.

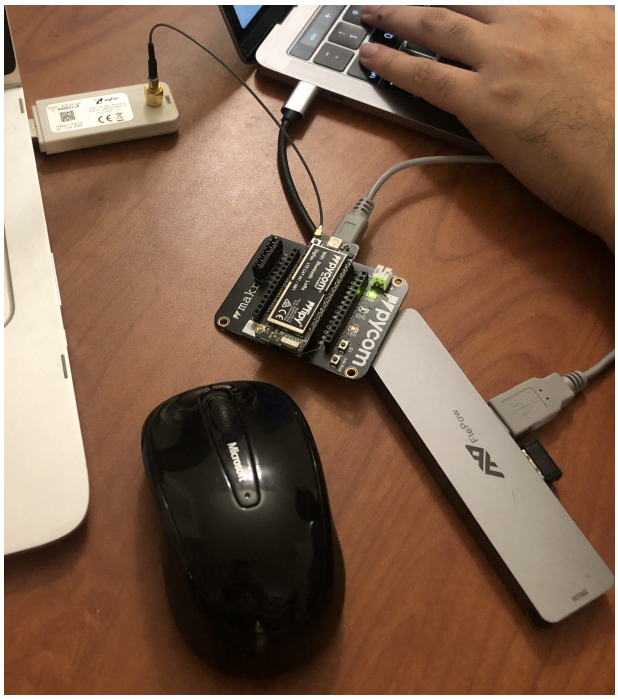

Fig. 4. The SigFox SDR Dongle (top left) and Pycom FiPy (middle) testbed

As mentioned earlier, an important thing to note is that Sigfox uses Advanced Encryption Standard (AES) 128 public key encryption to perform authentication and data integrity checks [5]. Each Sigfox device has the option to enable or disable AES. Without authentication enabled, the SNE can still receive uplink messages but cannot authenticate the sender and thus is not able to issue downlink messages. Additionally, this authentication mechanism along with the public key must be manually embedded in the packet within the client script.

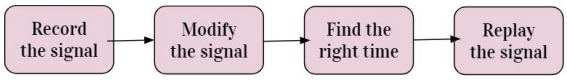

Fig. 5. Procedure Graph for launching the MITM attack

3.2 Record and replay attack

The breakdown of steps that are needed to launch the attack is outlined below (also shown in Figure 5):

- Recording the signal: To launch a replay attack, we first need the raw signal that is sent by the client node
- Modifying the signal: A malicious attacker will most likely need to modify the captured signal to cause harm to the base station or the Sigfox network. Hence, for this step we first attempt to correctly decode the signal/data packet. Once decoded, it is easier to change the data payload and/or the sequence number and then re-encode the signal so that it is successfully accepted by the network.
- Timing of the replay: Simply replaying this malicious signal at any time will not be sufficient to launch an attack because any out-of-order sequence number will be dropped by the network. Hence, we need to wait for the next cycle of sequence number (post counter reset) to replay our modified signal.
- Replaying the signal: Once we identify the appropriate slot (or time, in this case), we should transmit this signal using the USRP's transmitter

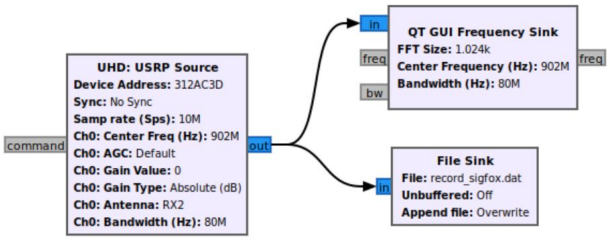

Fig. 6. GNU Radio flow graph for recording the signal

3.3 Launching the attack

To demonstrate the MITM attack, we use USRP B210 to emulate the attacker node. Being the most widely used Software Defined Radio (SDR) owing to its high performance and wide frequency range (70Mhz to 6GHz), it is an industry-ready platform. To program the USRP, we make use of GNU Radio Companion [2] - an open-source SDR-programming software. The main benefit of the GUI comes from its easy plug-and-play and visualization environment.

Figure 6 outlines the flow graph we created to capture Sigfox communication between FiPy and the Base Station. The USRP Source module opens the device's receiver at the center frequency of 902MHz the output is then fed into the frequency and file sink. We visualize the captured signals using the frequency sink and use the file sink module to store the signals in a .dat file. To replay the signal, we created the flow graph as seen in Figure 7. Here, the recorded signal (Figure 10) is being consumed by the USRP sink which is transmitting it at the center frequency of 902MHz. We are able to simultaneously visualize the transmission using the frequency sink module.

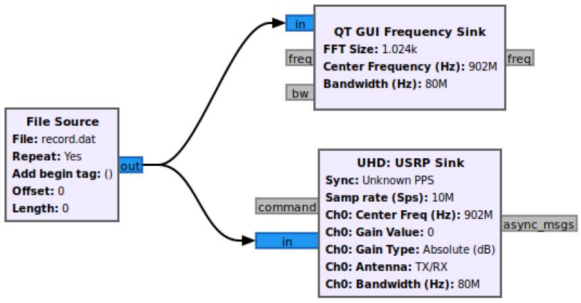

Fig. 7. GNU Radio flow graph for replaying the signal

4 OVERCOMING TECHNICAL OBSTACLES

We were able to communicate successfully between the FiPy and the Sigfox Base Station over the air. Additonally, we found that the Sigfox works quicker outside rather than inside buildings and have always been able to get the uplink working properly (i.e. receiving bytes from the FiPy on the Sigfox base station). We have not been able to get the downlink to work locally. However, using the Sigfox Backend instead of a local callback, we were able to successfully receive the downlink on the FiPy's side.

We used our SDR dongle to create a communication with the FiPy device. Once the message is received by the SDR dongle, Sigfox creates a downlink message to the FiPy using a callback function. We set up our Sigfox to return a simple 8 Byte message "CAFEBABE" using a local host web application. The Sigfox backend software connects to our host and sends a payload to which we reply with a HTTP 200 OK response with the payload we need or a HTTP 204 No Content response with no message sent back.

The FiPy device does not have a encryption state and does not share a public key with the Sigfox base station as stated on their

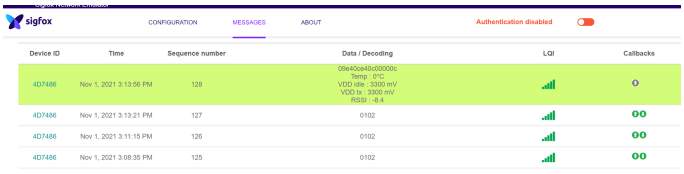

Fig. 8. Successful downlink response (highlighted in green) from the Sigfox Backend

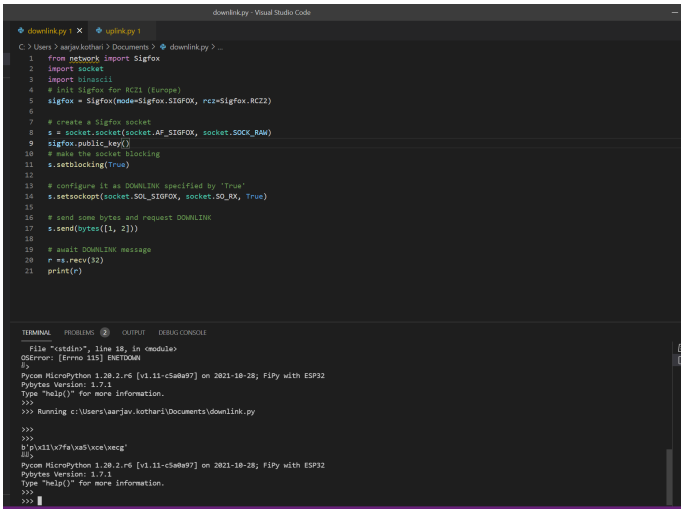

Fig. 9. Code snippet of python script running on Pycom FiPy to attempt to communicate bidirectionally with the Sigfox SDR Dongle

website [3] [5]. This leads to unstable downlink connections. We have had issues where our message is acknowledged by the Sigfox base station but not retrieved by our FiPy. Unfortunately, we were not able to identify the possible reasons as to why we sporadically could not receive the uplink. Sigfox documentation suggests that there is a downlink limit of 4 messages per day which also could be the reason that we have inconsistent results for the downlink [5].

Moreover, we have faced challenges in our attempts to capture the Sigfox communication between the FiPy and the Sigfox base station. The USRP device was not being recognized by the UHD libraries of the GNU Radio which prevented us from accessing and programming the radio on the SDR. We also had issues setting up GNU Radio on our virtual machines as we could not connect the USRP to our virtual machines.

We decided to divide our tasks and split the team into one team working on figuring out how to get USRP board set up on the lab machine while the second team worked on figuring out how to get the environment set up on the virtual machines. We successfully got both these tasks solved and could continue onward to our network attack strategies.

Our initial strategy to figure out a network attack was a simple replay attack. We were directly going to send the signal from the USRP board and let the base station capture it. For this, our main roadblock was to figure out how to successfully listen and decode the sigfox signal. We faced many challenges in doing so starting from the USRP to being unable to demodulate the signal.

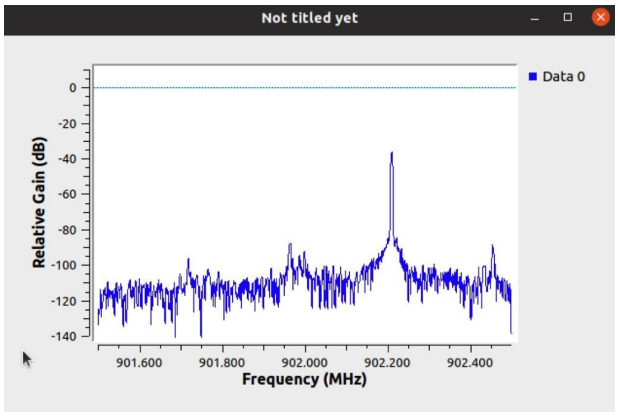

Fig. 10. Captured and recorded Sigfox signal

We recorded our signal (see peak on Figure 10) on the USRP generated from the PyCom device using GNU Radio. We recorded this signal into a DAT file. We tried to replay this signal but the base station could not detect it. We tried to improve the power gain on the signal but still could not capture it on the base station. Our initial thought was that the signal is being dropped by the base station due to authentication. We re-recorded the signal with the authentication turned off to also not get any result. Along with this, we disengaged the sequence number, which also did not work.

We decided to decode the signal to understand the preamble and the message and essentially ensure that the data recorded by the USRP was correct. As the signal system is not open source, we had to rely on external sources to emulate the results.

Due to the failure of the above method, we had to decode the message to understand the causation of the failure. We were trying to figure out if the messages were being dropped by the base station or was the signal incorrect during transmission from the USRP. To counter this, our first approach was to use the Renard-Phy reverse engineered Sigfox protocol library that was advertised to successfully demodulate uplink and downlink signals. The Renard-Phy source consisted of a Python script that would take a Sigfox WAV file as an input and output the payload along with the device characteristics. We used the demo files provided by the source and

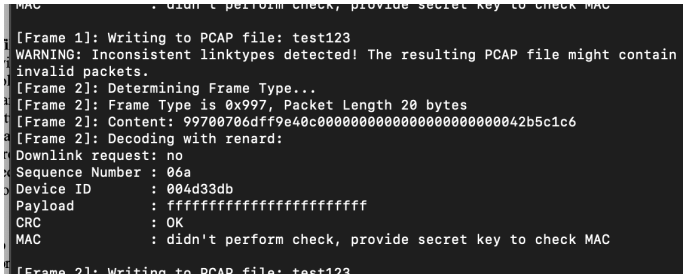

Fig. 11. Snippet of the result found from Renard-Phy on a demo file

it seemed very helpful to understand the packet and maybe even change its content if required. This protocol is very easy to use and can quickly demodulate for small file sizes. However, having the input set at WAV file is a constraint and capturing the radio signal in the required format is hard.

We could not capitalize on this protocol as we failed to correctly convert the signal file to a WAV file required. We tried it out with a few python scripts and a few multimedia software to only receive a static signal which could not be deciphered.

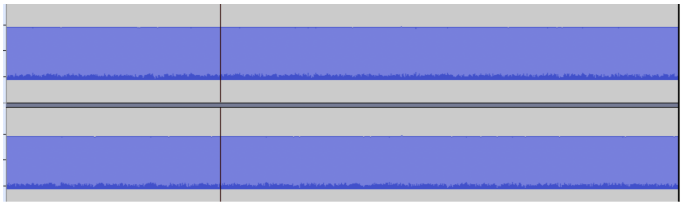

Fig. 12. Able to generate only static WAV file of the Sigfox recorded signal

Our second approach was to use the Sigfox Radio Analyzer (RSA) [4] created by the Sigfox team themselves. The tool is very powerful as designed and can be used to test modulation, demodulation, and visualize radio-frequency measurement results. We believe that this should be the full proof method of demodulating because it is a propriety software and it should have the necessary functionality to decode any Sigfox signal in any input, whether it's a direct network signal or passing a recorded signal. The only pain point of this software was that the RSA is on an ISO disk that needs to be booted from a USB on a machine and cannot be installed directly. This removes the flexibility of using the software anywhere and everywhere and we were stuck with a standalone device that can boot the ISO.

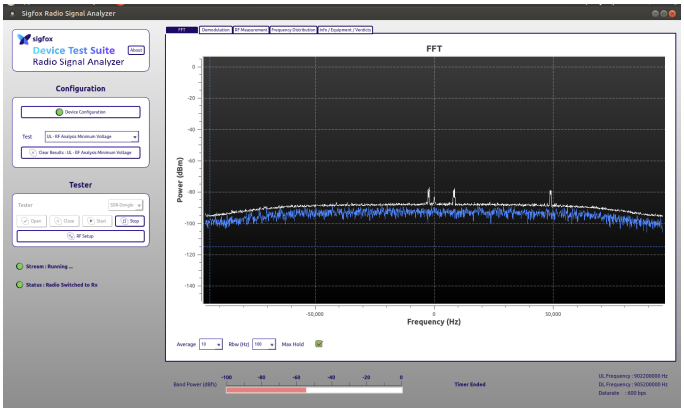

Fig. 13. RSA capturing our replayed signal

We successfully could read the raw frames of our messages sent from the FiPy. We could decode the message and understand the different packet components of the signal. However, we could not decode the signal replayed from the USRP. We tested out multiple methods along with increasing power gain but the RSA would not demodulate the headers. Upon replay, we did notice peaks on the signal analyzer window of RSA but could not access the raw frames.

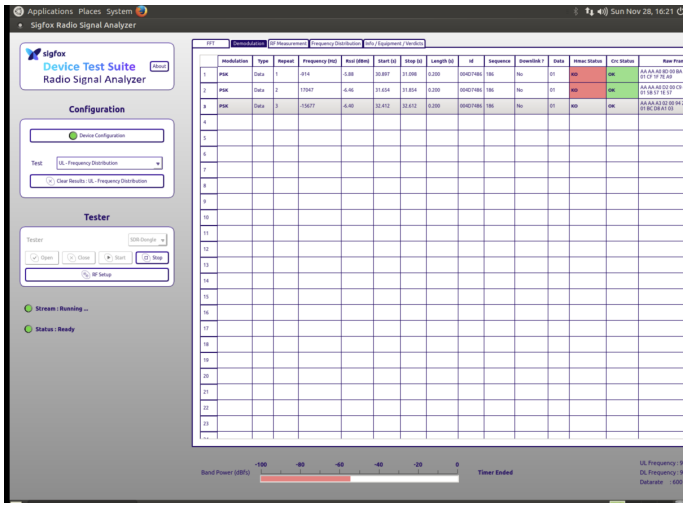

Fig. 14. RSA successfully decoding the headers of the message sent from the FiPy

5 EVALUATION

Through our work on this project, our team was able to fulfill the first objective of the project description which was to successfully setup the uplink connection between the Pycomm FyPi device and the Sigfox base station. The next objective was to setup a client application that accepts and reads the data from the Sigfox base station, which we were able to accomplish using the downlink and callback functionality with both our local application and the Sigfox backend. The last objective was to write an SDR program with USRP to capture and replay the data. Using the USRP, our team was able to intercept this communication between the two devices and capture the format as a .dat file. We attempted to replay the attack but were unable to capture this replay on the Sigfox base station device or the FyPi.

6 CONCLUSION AND FUTURE DIRECTIONS

Ultimately, the motivation for this project was to explore Sigfox communication and understand how vulnerable it is to man-in-the-middle/replay attacks. As a team, we certainly performed the hard work to setup and intercept the communications between devices using the Sigfox network and were able to capture and attempt replays of the captured signal. We were not able to successfully retrieve this replay on the Sigfox base station itself.

The next steps for this project would be to understand the underlying fundamentals of the Sigfox signal. From the software perspective, tweaking the signal a little by the way of amplifying, modifying or decoding could help us successfully replay it. In addition to this, we could also look at updating the hardware or the antenna to achieve better results.

7 ACKNOWLEDGEMENTS

We would like to extend our gratitude to Prof. Songwu Lu for giving us the opportunity to work on this project and our mentor, Mr. Zhaowei Tan for his continuous support and encouragement.

REFERENCES

[1] [n.d.]. CLOUD.GOOGLE.COM. https://cloud.google.com/community/tutorials/sigfox-gw. Accessed: 2021-10-21.

[2] [n.d.]. GNURADIO.COM. https://www.ettus.com/sdr-software/gnu-radio. Accessed: 2021-10-15.

[3] [n.d.]. PYCOM.COM. https://pycom.io/product/fipy. Accessed: 2021-09-15.

[4] [n.d.]. Radio Signal Analyzer. https://support.sigfox.com/docs/radio-signal-analyser-user-guide. Accessed: 2021-09-15.

[5] [n.d.]. SIGFOX.COM. https://www.sigfox.com. Accessed: 2021-09-15.

[6] [n.d.]. USRP B210. https://www.ettus.com/all-products/ub210-kit/. Accessed: 2021-09-15.

[7] Florian Laurentiu Coman, Krzysztof Mateusz Malarski, Martin Nordal Petersen, and Sarah Ruepp. 2019. Security Issues in Internet of Things: Vulnerability Analysis of LoRaWAN, Sigfox and NB-IoT. In *2019 Global IoT Summit (GIoTS)*. 1–6. https://doi.org/10.1109/GIOTS.2019.8766430