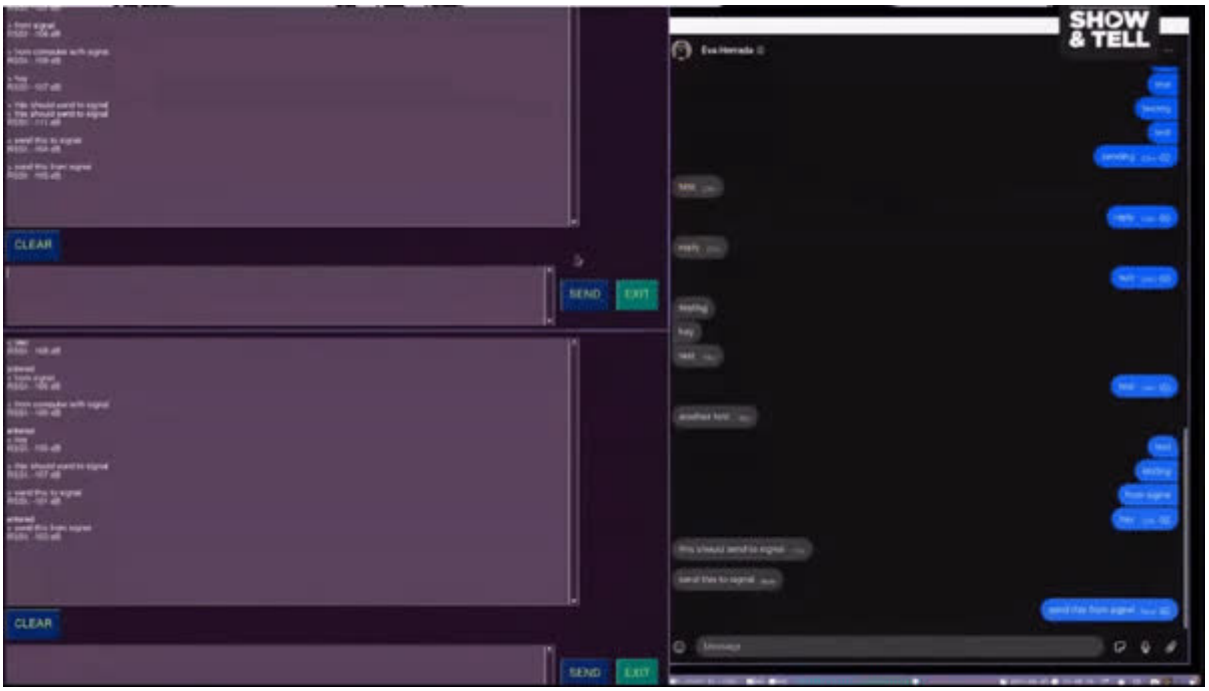
LoRa Signal Bridge with the Feather RP2040 RFM

Created by Eva Herrada



https://learn.adafruit.com/lora-signal-bridge-with-the-feather-rp2040-rfm

Last updated on 2024-06-03 03:51:44 PM EDT

Table of Contents

Overview

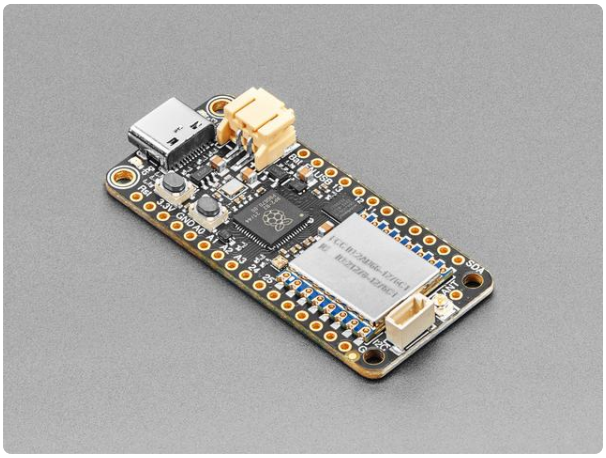
The goal with this project is to demonstrate a way to add another layer of distance between two people messaging on Signal by using LoRa radios as a transport mechanism. It's also a good demo of how to use our Feathers as 'peripherals' for a desktop computer. We also show how to send messages encrypted with AES (but without including who the sender or receiver are).

This project is intended to be a showcase of one way to do communication without the use of pre-existing infrastructure. It could in theory be easily modified to work as a mesh network, although the protocol would have to be implemented on top of LoRa and is outside the scope of this guide

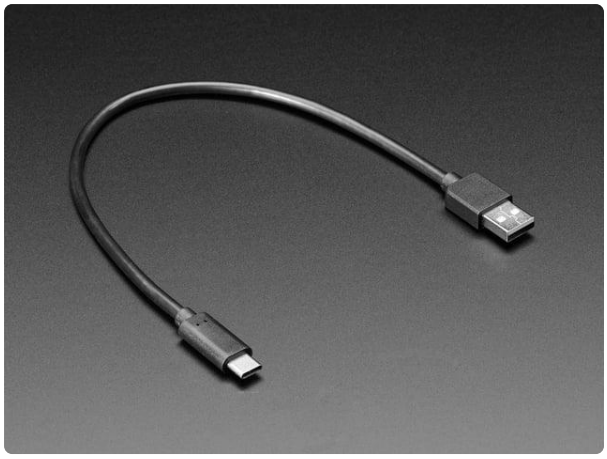
Depending on your location and frequency bands, encrypted communications are not permitted for radio messages. LoRa radios as they broadcast on non-communication ISM bands, but be sure to research your local regulations pertaining to radio communication before doing this project.

Materials

You will need 2 of each of these. This project will also work with any other RFM95 board or FeatherWing provided you update the pins.



Adafruit Feather RP2040 with RFM95 LoRa Radio - 915MHz
This is the Adafruit Feather RP2040 RF95 LoRa Radio. We call these RadioFruits, our take on a microcontroller with a...
https://www.adafruit.com/product/5714



USB Type A to Type C Cable - 1ft - 0.3 meter

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

https://www.adafruit.com/product/4473

CircuitPython

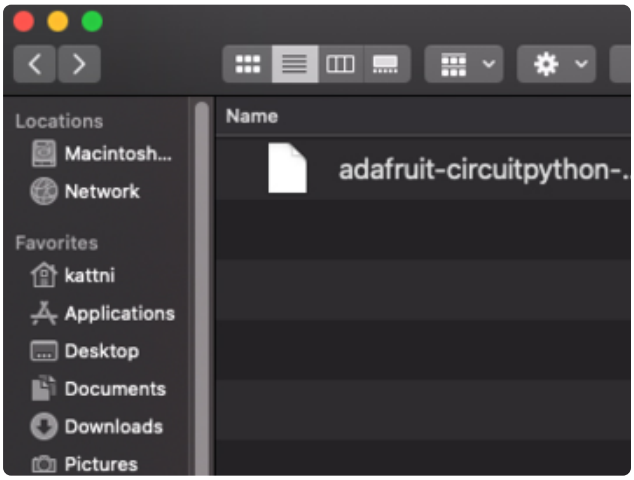
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

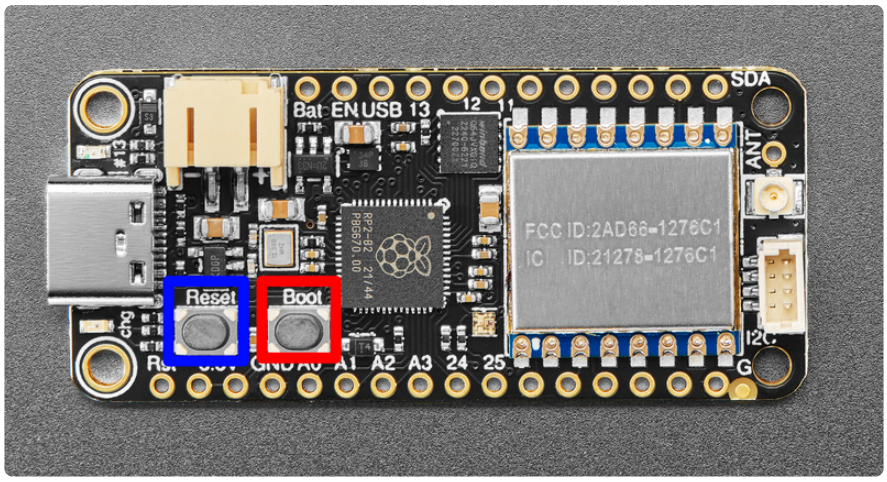
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/18Co



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

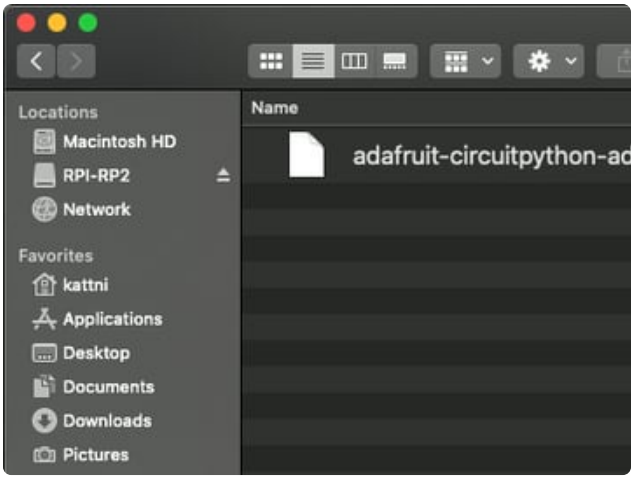


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

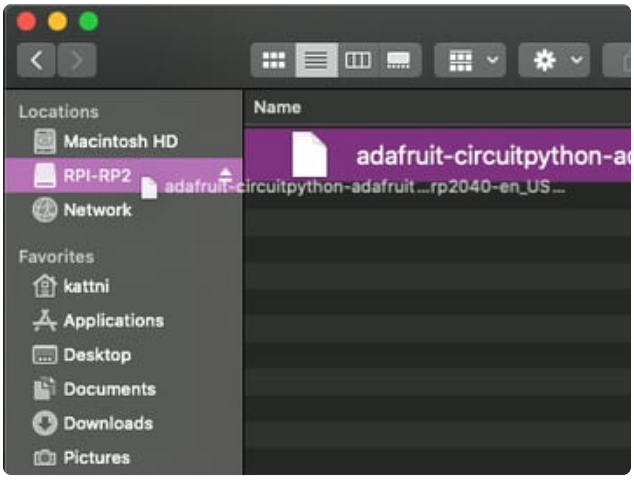
If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.

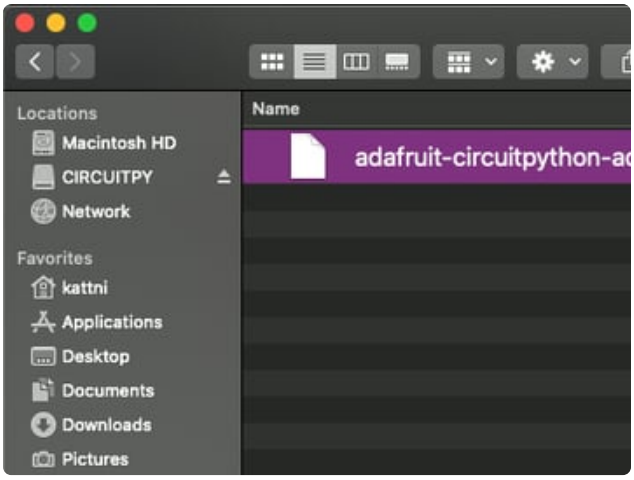
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **RPI-RP2**.



Drag the **adafruit_circuitpython_etc.uf2** file to **RPI-RP2.**



The **RPI-RP2** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it, you're done! :)

Safe Mode

You want to edit your **code.py** or modify the files on your **CIRCUITPY** drive, but find that you can't. Perhaps your board has gotten into a state where **CIRCUITPY** is read-only. You may have turned off the **CIRCUITPY** drive altogether. Whatever the reason, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in **boot.py** (where you can set **CIRCUITPY** read-only or turn it off completely). Second, it does not run the code in **code.py**. And finally, it does not automatically soft-reload when data is written to the **CIRCUITPY** drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the **CIRCUITPY** drive.

Entering Safe Mode

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

If you successfully enter safe mode on CircuitPython, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot.
Press again to exit safe mode.

Press any key to enter the REPL. Use CTRL-D to reload.
```

You can now edit the contents of the **CIRCUITPY** drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.

Flash Resetting UF2

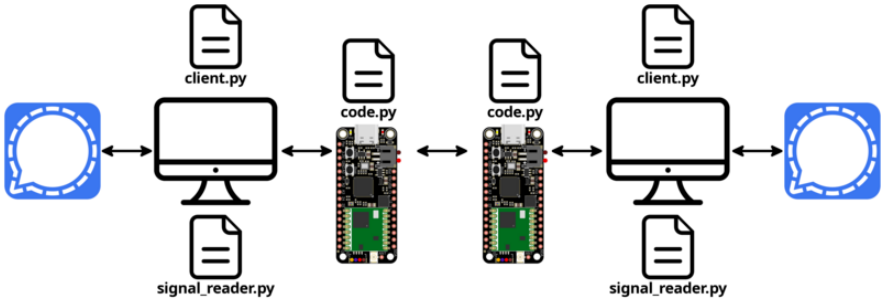
If your board ever gets into a really weird state and CIRCUITPY doesn't show up as a disk drive after installing CircuitPython, try loading this 'nuke' UF2 to RPI-RP2. which

will do a 'deep clean' on your Flash Memory. **You will lose all the files on the board**, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

Download flash erasing "nuke" UF2

https://adafru.it/RLE

Setting up your computer



This project has only been tested in Linux. It may work on other operating systems but would likely require some modification to do so.

Setting up signal-cli

This step isn't necessary for the project to work, but if you want to integrate it into signal you will need to do it.

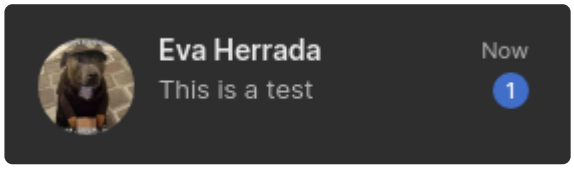
Install signal-cli with the instructions here

https://adafru.it/18Fn

After that, go down to the usage section and register your account.

Register this Signal account with a number different from your own. Using a Google Voice number is a good way to do this (after registration of the Signal account, you can deregister the Google Voice number and create a new one to make as many Signal accounts as you want).

Test it by sending a message to yourself or someone else. Make sure to include the country code (https://adafru.it/18Fr).



Installing Python libraries

Install the following libraries with pip:

```
pip3 install PySimpleGUI
```

```
pip3 install pyserial
```

```
pip3 install cryptography
```

If these libraries installed correctly, you should be all set!

Code the RP2040 radio

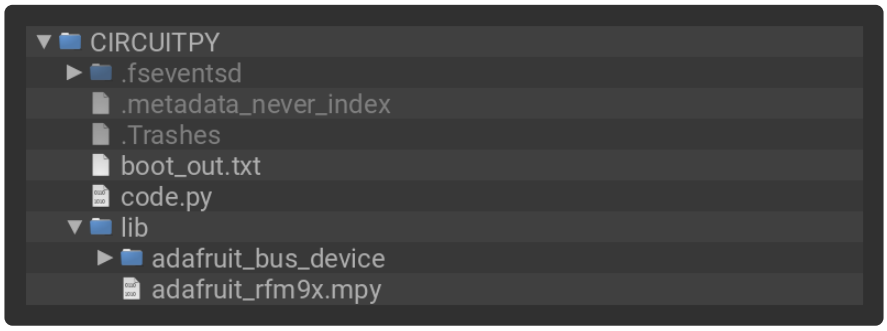
Installing Project Code

To use with CircuitPython, you need to first install a few libraries into the **lib** folder on the **CIRCUITPY** drive which appears on your computer when the Feather board is plugged in via USB. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **RP2040_Radio_Messenger/** and then click on the directory that matches the version of CircuitPython you're using.

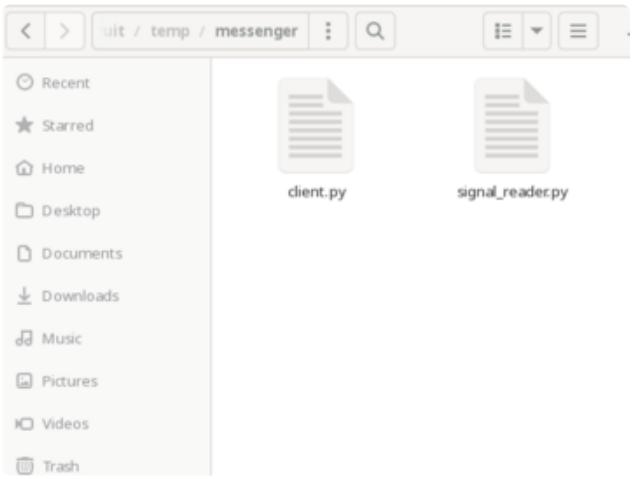
Connect your RP2040 board to your computer via a known good USB data+power cable. The board should show up as a thumb drive named **CIRCUITPY** in Explorer or Finder (depending on your operating system). Copy **code.py** and the **lib/** directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:




```
# SPDX-FileCopyrightText: 2023 Eva Herrada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import random
import board
import usb_cdc
import digitalio

import adafruit_rfm9x

spi = board.SPI()

# radio setup
RADIO_FREQ_MHZ = 915.0

LED = digitalio.DigitalInOut(board.LED)
LED.direction = digitalio.Direction.OUTPUT

CS = digitalio.DigitalInOut(board.RFM_CS)
RESET = digitalio.DigitalInOut(board.RFM_RST)

rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, RADIO_FREQ_MHZ)

rfm9x.tx_power = 23

# Wait to receive packets.  Note that this library can't receive data at a fast
# rate, in fact it can only receive and process one 252 byte packet at a time.
# This means you should only use this for low bandwidth scenarios, like sending
# and receiving a single message at a time.
print("Waiting for packets...")

MESSAGE = b""
ID = None
while True:
    char = usb_cdc.console.read(usb_cdc.console.in_waiting)
    if char:
        MESSAGE += char
        # print(char.decode('utf-8'), end="")
        if char[-1:] == b"\r":
            MESSAGE = MESSAGE[:-1]
            ID = random.randint(0, 1000)
            rfm9x.send(bytes(f"{ID}|", "utf-8") + MESSAGE)
            print(f"{ID}|{MESSAGE.decode()}")
```

```
            timestamp = time.monotonic()
            sent = MESSAGE
            MESSAGE = b""
            continue

    packet = rfm9x.receive()

    if packet is None:
        # Packet has not been received
        LED.value = False
    else:
        # Received a packet!
        LED.value = True

        try:
            PACKET_TEXT = str(packet, "ascii")
        except UnicodeError:
            print("error")
            continue

        print(PACKET_TEXT)
        mess_id, text = PACKET_TEXT.split("|")
        if mess_id != "-1":
            rfm9x.send(bytes(f"-1|{mess_id}", "utf-8"))
            print(f"Received: {PACKET_TEXT}")
        else:
            print(f"Delivered: {text}")
            ID = None

        rssi = rfm9x.last_rssi
        print(f"RSSI: {rssi} dB")
```



Next, create a directory on your computer and place
both **client.py** and **signal_reader.py** in it

Now, go into **client.py** and edit the `NUMBER` variable to the phone number associated with your Signal account connected to a GUI.

Code run-through

code.py run-through

First, the code imports all the required libraries.

```
import time
import random
import board
import usb_cdc
import digitalio

import adafruit_rfm9x
```

Next, the radio is set up, if you would like to use this with a board other than the RFM RP2040, change the pins that `CS` and `RESET` use.

```
# radio setup
RADIO_FREQ_MHZ = 915.0

LED = digitalio.DigitalInOut(board.LED)
LED.direction = digitalio.Direction.OUTPUT

CS = digitalio.DigitalInOut(board.RFM_CS)
RESET = digitalio.DigitalInOut(board.RFM_RST)

rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, RADIO_FREQ_MHZ)

rfm9x.tx_power = 23
```

The code now initializes a few important variables before entering the main loop.

```
# Wait to receive packets.  Note that this library can't receive data at a fast
# rate, in fact it can only receive and process one 252 byte packet at a time.
# This means you should only use this for low bandwidth scenarios, like sending
# and receiving a single message at a time.
print("Waiting for packets...")

MESSAGE = b""
ID = None
```

In the first part of the main loop, the code checks to see if it has received any messages over the serial port. If it has, it forwards the message to the RFM to be sent over radio.

```
while True:
    char = usb_cdc.console.read(usb_cdc.console.in_waiting)
    if char:
        MESSAGE += char
        # print(char.decode('utf-8'), end="")
        if char[-1:] == b"\r":
            MESSAGE = MESSAGE[:-1]
            ID = random.randint(0, 1000)
```

```
            rfm9x.send(bytes(f"{ID}|", "utf-8") + MESSAGE)
            print(f"{ID}|{MESSAGE.decode()}")
            timestamp = time.monotonic()
            sent = MESSAGE
            MESSAGE = b""
            continue
```

The last bit of code in this file handles what happens when a message is received and sending a reply to a received message to confirm receipt. It unpacks the message and sends it over serial along with a bit of other information about the signal strength.

```
    packet = rfm9x.receive()

    if packet is None:
        # Packet has not been received
        LED.value = False
    else:
        # Received a packet!
        LED.value = True

        try:
            PACKET_TEXT = str(packet, "ascii")
        except UnicodeError:
            print("error")
            continue

        print(PACKET_TEXT)
        mess_id, text = PACKET_TEXT.split("|")
        if mess_id != "-1":
            rfm9x.send(bytes(f"-1|{mess_id}", "utf-8"))
            print(f"Received: {PACKET_TEXT}")
        else:
            print(f"Delivered: {text}")
            ID = None

        rssi = rfm9x.last_rssi
        print(f"RSSI: {rssi} dB")
```

client.py run-through

This file is one of the two files that runs on your computer. It starts out by importing all the necessary libraries.

```
import time
import pathlib
import os

import PySimpleGUI as sg
import serial
from cryptography.fernet import Fernet
```

Then, the phone number to send to is set and the secret key is either created or read.

```
# Replace NUMBER with your phone number including country code
NUMBER = "REPLACE_ME"

try:
    with open("secret.key", "rb") as key_file:
```

```
        key = key_file.read()
except FileNotFoundError:
    with open("secret.key", "wb") as key_file:
        key_file.write(Fernet.generate_key())

    with open("secret.key", "rb") as key_file:
        key = key_file.read()
```

Before it goes into the main function, there are a few last variables to intialize.

```
aes = Fernet(key)

port = int(input("port: /dev/ttyACM"))

s = int(input("signal?"))

ser = serial.Serial(f"/dev/ttyACM{port}", 115200, timeout=0.050)

last_time = time.monotonic()
```

Then, the main function starts, the code sets up the GUI before initializing some important variables before the main loop starts

This file is a modified version of a PySimpleGUI example. See the original by clicking this button

https://adafru.it/18Ft

```
def ChatBotWithHistory():  # pylint: disable=too-many-statements,too-many-
branches,too-many-locals
    # -------  Make a new Window  ------- #
    # give our form a spiffy set of colors
    sg.theme("DarkPurple4")

    layout = [
        [sg.Text("Encrypted LoRa Messaging Client", size=(40, 1))],
        [sg.Output(size=(127, 30), key="history", font=("Helvetica 10"))],
        [sg.Button("CLEAR", button_color=(sg.YELLOWS[0], sg.BLUES[0]))],
        [
            sg.ML(size=(85, 5), enter_submits=True, key="query",
do_not_clear=False),
            sg.Button(
                "SEND", button_color=(sg.YELLOWS[0], sg.BLUES[0]),
bind_return_key=True
            ),
            sg.Button("EXIT", button_color=(sg.YELLOWS[0], sg.GREENS[0])),
        ],
    ]

    window = sg.Window(
        "Chat window with history",
        layout,
        default_element_size=(30, 2),
        font=("Helvetica", " 13"),
        default_button_element_size=(8, 2),
        return_keyboard_events=True,
    )

    # ---===--- Loop taking in user input and using it  --- #
    command_history = []
```

```
history_offset = 0

last_id = None
```

In the main loop, the code first tries to read from the window. Then it tries reading from the serial port and forwards the message along to Signal.

```
while True:
    event, value = window.read(timeout=0.05)

    while ser.in_waiting:
        data_in = ser.readline()
        mess = data_in.decode()[:-1]
        try:
            if mess.startswith("Received"):
                mess = mess.split(" ", 1)[1]
                mess_id, text = mess.split("|", 1)
                text = aes.decrypt(text.encode()).decode()
                print(f"&gt; {text}")
                if s:
                    os.system(f'signal-cli send -m "{text}" {NUMBER}')
            elif "|" in mess:
                mess_id, text = mess.split("|", 1)
                last_id = int(mess_id)
            if "Delivered" in mess:
                mess_id = int(mess.split(" ", 1)[1])
                if mess_id == last_id:
                    print("Delivered")
            if "RSSI" in mess:
                print(mess + "\n")
        except Exception as error:  # pylint: disable=broad-except
            print(error)
```

This branch checks to see if a new message has been received to the Signal account and if it has been it displays it and sends it to the RP2040 via Serial.

```
       if os.path.isfile("message"):
        print("entered")
            time.sleep(0.1)
            path = pathlib.Path("message")
            with path.open() as file:
                signal = file.readline().rstrip()
            print(f"< {signal}")
            ser.write(aes.encrypt(signal.encode()))
            ser.write(signal.encode())
            ser.write("\r".encode())
            command_history.append(signal)
            history_offset = len(command_history) - 1
            # manually clear input because keyboard events blocks clear
            window["query"].update("")
            os.system("rm message")
```

This branch handles what to do when certain window inputs are received. If a message is to be sent directly from the window, it encrypts it and sends it, if the user has hit the 'Exit' button, it closes the window, and if the user navigates up or down the chat window, it moves the window accordingly.

```
        if event:
            if event == "SEND":
```

```
                query = value["query"].rstrip()
                print(f"&lt; {query}")
                ser.write(aes.encrypt(query.encode()))
                ser.write(query.encode())
                ser.write("\r".encode())
                command_history.append(query)
                history_offset = len(command_history) - 1
                # manually clear input because keyboard events blocks clear
                window["query"].update("")
                # EXECUTE YOUR COMMAND HERE

            elif event in (sg.WIN_CLOSED, "EXIT"):  # quit if exit event or X
                break

            elif "Up" in event and len(command_history) != 0:
                command = command_history[history_offset]
                # decrement is not zero
                history_offset -= 1 * (history_offset &gt; 0)
                window["query"].update(command)

            elif "Down" in event and len(command_history) != 0:
                # increment up to end of list
                history_offset += 1 * (history_offset &lt; len(command_history) - 1)
                command = command_history[history_offset]
                window["query"].update(command)

            elif event == "CLEAR":
                window["history"].update("")

            elif "Escape" in event:
                window["query"].update("")
```

Finally, the above function is run.

```
ChatBotWithHistory()
```

signal_reader.py run-through

This is a helper file specifically to check signal for new messages in a non-blocking way. It starts by importing the required libraries.

```
from subprocess import check_output
import os
```

The code then enters the main loop, starting by checking if there are new messages from Signal. If there are, it formats them in a more desirable way and writes them to a file, named **message**.

```
while True:
    SIGNAL = check_output('signal-cli receive', shell=True).decode("utf-8")
    if "Body" in SIGNAL:
        signal_msg = SIGNAL.split("Body")[1].split("\n")[0][2:]
        print(signal_msg)
        os.system("touch message")
        with open("message", "w") as F:
            F.write(signal_msg)
```

Using the Radio Messenger

Here's a quick demonstration of me using the radio messenger. I have one messenger connected to Signal and one not. Both boards are plugged into the same computer.

Using with Signal

To use with Signal, start by opening two terminal windows and navigating to the folder you're storing **client.py** and **signal_reader.py** in.

run `python3 signal_reader.py`

Send yourself a message to verify it's working correctly, it should show up in a few seconds.

Using without Signal

If you did not do the previous step, simply open a terminal window and navigate to where **client.py** is stored. You do not need to run **signal_reader.py**.

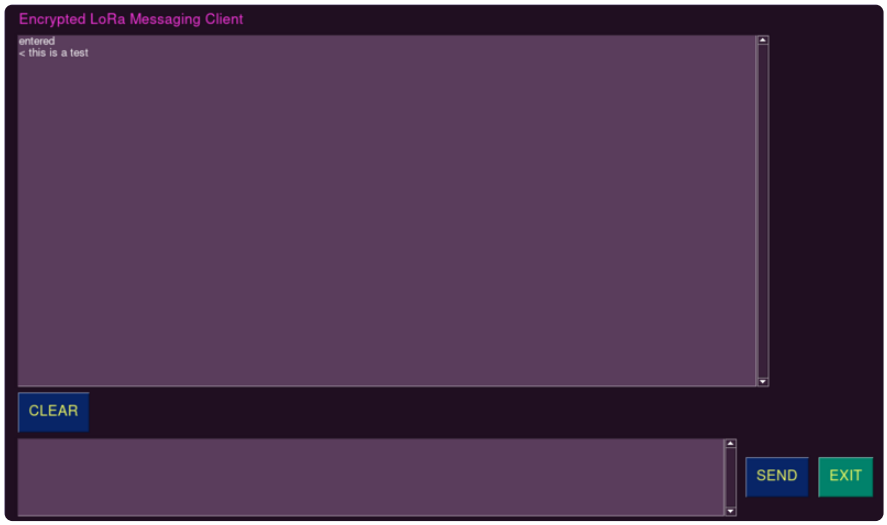
Starting the messenger

Now, run `python3 client.py`

You'll see an option for the port, type which number the port is (on Linux if you only have one device plugged in, just typing in **0** and pressing enter should work).

Choose if you want to enable signal (**1** for yes **0** for no) or not and then press enter.

A window should now pop up and the test message you sent earlier (if you enabled signal) should appear.



Now, copy the newly created **secret.key** over to your recipient and you should be good to go!