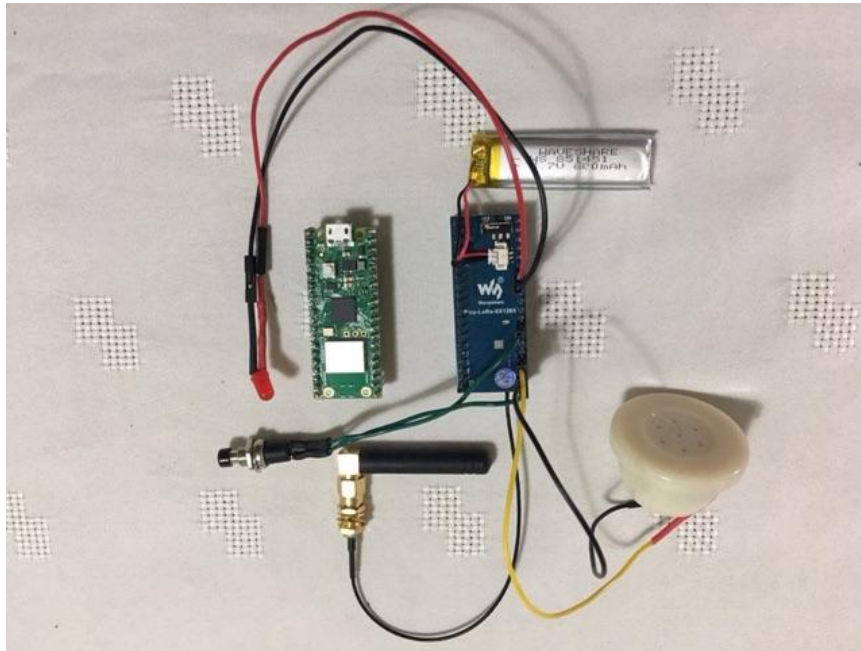


# LoRa P2P Wireless Gate Alarm

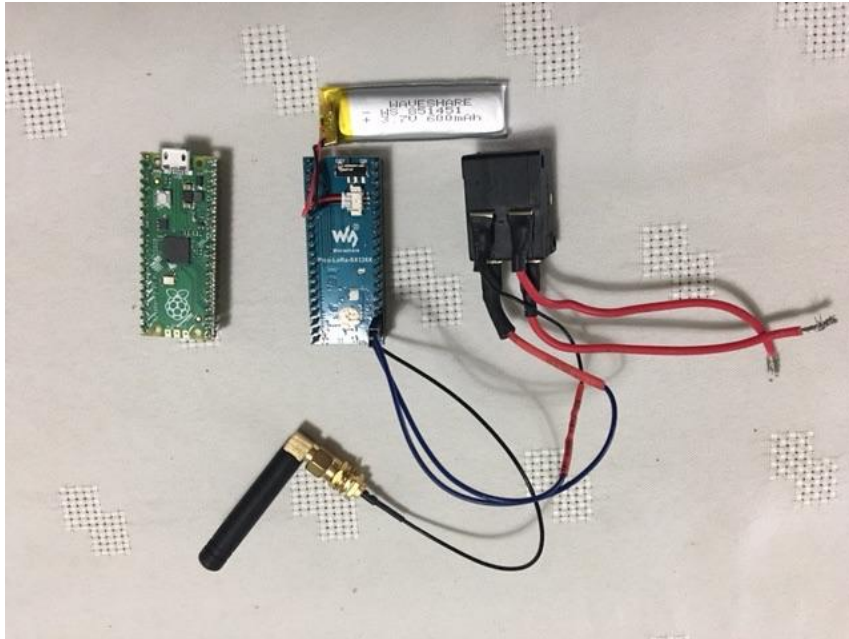
## Main House Device



Mounted in an old Pi case  
(Pico in a Pi case)



## Main Gate Device



## Solar Panel

Pico included for scale



# Introduction

**My front gate is a long way from the house at around 300m. I don't want people wandering around my property without knowing about it. This project uses two Raspberry Pi Pico's and two PiicoDev LoRa modules. One standard Pico is at the gate and the other is a wifi model which is at my house. When the gate is opened a micro switch is triggered and a text signal is sent via a LoRa radio signal to the house Pico. The house device plays a tone and writes to a log file. I decided to write the software using modular techniques so that any or all of it could be easily extracted for other projects or so that this project could be easily extended.**

## Main Project Goals

Detect when the Front Gate is opened and play a sound inside the house.  
Create modular software that can be reused on other projects.

## Requirements

- Gate end is solar powered as no power is available.
- Budget is ~ \$100.00 dollars AU
- Log details of Gate Open events on the house device
- Indicate if the connection between the house and the gate has timed out by an error LED on the house device.
- Indicate number of gate open events on the house device LED since last reset
- Log battery levels of the gate device on the house device
- Indicate if the battery power level at the gate is low by a warning LED on the house device
- Publish the house device log file via a mini web server on the house device
- Set the Date Time on the house device via WIFI connection to a SNTP time server
- Set the Date Time on the gate device via a LoRa message from the house device
- Encrypt / Decrypt messages sent between devices
- Allow easy addition of extra devices and message fields

## Possible Project Extensions

- Water Tank Monitoring and Logging
- Dam Level Monitoring and Logging
- Wind Speed Monitoring and Logging
- Rainfall Monitoring and Logging
- Temperature Monitoring and Logging

- Anything you can measure with a Pico

Each of these ideas could be implemented on a remote device and report back and be logged on the central unit. Controls and adjustments could be implemented via the web interface. Some devices could fulfill multiple roles with messages easily passed back and forward to named devices and ignored on others.

## Hardware



### Main House




#### Pin Connections

LED connected to physical Pin 7(GPIO 5) and physical pin 8 (Gnd)

Speaker connected physical Pin 17 (GPIO 13) and physical pin 18 (Gnd)

#### Parts

Qty	Device	Link and Description
1	Raspberry Pi Pico With WiFi 	<a href="#">Raspberry Pi Pico W with Soldered Male Headers   Core Electronics Australia</a>
1	Lora Module 	<a href="#">SX1262 LoRa Node Module for Raspberry Pi Pico. LoRaWAN   Waveshare   Core Electronics Australia</a> Identical to the one used on the Main Gate Device. This device comes with a battery and an antenna. It also has a built-in charging circuit and will charge the battery from the pico usb port.
1	Speaker	I salvaged a speaker from a telephone handset.

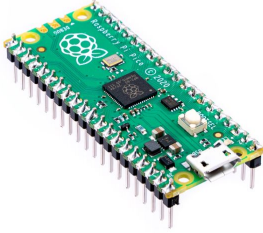
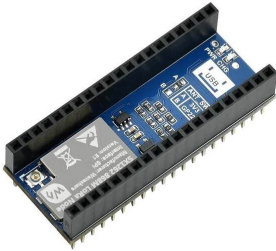
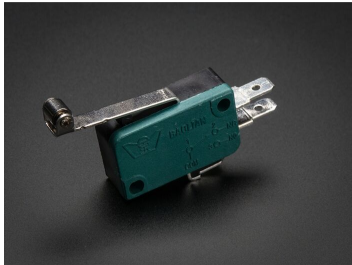
		<p>I reckon one of these would be perfect  <a href="#">Piezo Buzzer - PS1240</a>   <a href="#">Adafruit ADA160</a>   <a href="#">Core Electronics Australia</a></p>
1	<p>LED</p> 	<p>I had one in stock and simply soldered a 200 ohm resistor onto one leg then covered it with a piece of heat shrink  <a href="#">LED Rainbow Pack - 5mm PTH</a>   <a href="#">Sparkfun COM-12903</a>   <a href="#">Core Electronics Australia</a></p>
1	<p>Push Button</p> 	<p>Momentary Switch normally off.  I had one in stock but it looks like this  <a href="#">Momentary Button - Panel Mount (Black)</a>   <a href="#">Sparkfun COM-11996</a>   <a href="#">Core Electronics Australia</a></p>
1	<p>Case</p>	<p>I just used an old Raspberry Pi 1 Model B Case I had lying around as the house unit is inside so it doesn't need to be water proof.</p>

# Main Gate


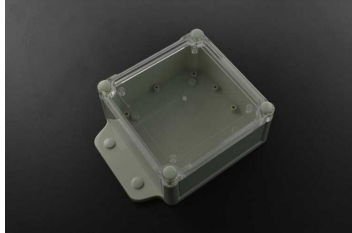
## Pin Connections

MicroSwitch connected to physical Pin 19(GPIO 14) and physical pin 18 (Gnd)

## Parts

Qty	Device	Link and Description
1	Raspberry Pi Pico 	Non wifi model <a href="#">Raspberry Pi Pico (with Soldered Headers)   Core Electronics Australia</a>
1	LoRa Module 	<a href="#">SX1262 LoRa Node Module for Raspberry Pi Pico. LoRaWAN   Waveshare   Core Electronics Australia</a> Identical to the one used on the Main House Device. This device comes with a battery and an antenna. It also has a built-in charging circuit and will charge the battery from the pico usb port.
1	Micro Switch 	I salvaged one out of a broken angle grinder but almost anything would do. The switch is normally open. The cam on the gate closes the switch. Something like this should work fine. <a href="#">Micro Switch w/Roller Lever - Three Terminal   Adafruit ADA819   Core Electronics Australia</a>



1	Solar Panel 	<p>I picked a usb phone charger solar panel on ebay. Beware power output claims are usually grossly exaggerated. Make sure it has a micro usb plug and output is 5v. The one I purchased is rated at 10w but I doubt it can actually produce that much power.</p> <p>**** Perhaps Core Electronics could recommend a solar panel here? ****</p>
1	Project Case 	<p><a href="#">Plastic Project Box Enclosure Waterproof Clear Cover - 6.61x4.72x 2.17 inch   DFRobot FIT0723   Core Electronics Australia</a></p>

## Process

I started off by sourcing all the different major components although I didn't build the cases until I knew what the finished dimensions of the components would be.

Note: The LoRa modules came with an antenna, small 600 mAh Lipo battery and charging circuit to power and charge from the Pico usb port. This saved the need to make a whole lot of decisions.

When programming, I always break things down into manageable hunks. I found or constructed demo code for each of the major functions. I installed each piece of the demo code in isolation to test and get that feature working. Then I debugged and adjusted the code to get the desired result. For example, how to play a sound or how to flash the LED in a pattern. I then incorporated each concept into the main program sharing code between modules where possible.

The tough one was the code for the LoRa modules. I lucked out and found a very detailed library that does exactly what I needed. Care needs to be taken here to make sure you meet the Radio Frequency requirements of your location. Even though the little things in this example are set up as peer to peer (P2P) their range is amazing over clear ground. (many kilometers)

## Range Test Results

350m	Success
1000m	Not Tested
2000m	Not Tested

Further than this around my house the hills get in the way. Long range stuff needs to be line of sight but unconfirmed reports indicate 3-15 km may be possible. Use the Ping Pong test found in the SX126 drivers folders to do a simple range test. Run the example on Device 1 and the LED on Device 2 will toggle when the devices are in range. The devices actually send the word ping and the reply is the word pong.

## Software

The full code for this project can be downloaded from this github repository

<https://github.com/pollardd/Lora-Gate-Alarm>

Some of the bleeding edge versions of Micropython had some very odd problems connecting to the pico once my programs became complicated. I believe the last stable version at the time of writing was working fine.

MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040

The most recent development version was also working.

Rp2-pico-20230209-unstable-v1.19.1-859-g41ed01f13.uf2

Micropython Interpreter downloads available here <https://micropython.org/download/rp2-pico/>

## Files Required

Common to both devices

Filename	Description
Sx1262.py	LoRa library (details below)
Sx126x.py	LoRa library (details below)
_sx126x.py	LoRa library (details below)
debug.py	Reusable debugging output code Output is to a file on the pico and to the serial console.
secrets.py	Contains Encryption/Decryption keys generated by “GenerateEncryptionKeys.py” This file is provided but not required on



	either device. Run the program once and share the keys on both devices.
constants.py	User modifiable constants for configuration
encryption.py	Contains code specific to the encryption / decryption process
loraMessage.py	Contains code specific to the transfer of messages between devices
mpyaes.py	Contains third party implementation of built in python encryption library
debugCounter.txt	Contains a counter for the debug unique identifier
debuglog.csv	Debugging log file automatically created

## House Device

Filename	Description
main.py	Renamed from mainHouse.py for the House Device to allow for automatic execution at startup
secretsHouse.py	Contains SSID and Password for your wifi network
dateTime.py	Formats date and time into your preferred format (used in debugging) currently configured for DD/MM/YYYY HH:MM:SS
ntpClientTZ.py	Third party code to retrieve the time from the internet using SNTP protocol.
Blink.py	Contains code specific to flashing a LED in a specific sequence to indicate gate open events or error messages
Subprocess.py	This file contains the code that runs as a second thread on processor 2.
tone.py	Specific code that plays a audible tone on the house device speaker
webServer.py	Code that displays a web page published on the local ip address (port 80).

	This page is currently only a proof of concept that allows the user to turn the LED on and off.
--	---

## Gate Device

Filename	Description
main.py	Renamed from mainGate.py to allow execution on startup

## Program Flow

The messages sent between the LoRa devices can be considered to be in plain text. Think of this like an old style serial / teletype connection where a character is pushed in one end of the cable and it pops out the other end. The way this is performed doesn't really matter to us, the application programmers. The magic is done within the hardware, software drivers and the LoRa communication module detailed below.

The LoRa connection and configuration is based on an example found here.

<https://github.com/ehong-tl/micropySX126X>

Here is a link to my initial Core Electronics Forum Question discussing the LoRa configuration options and country requirements. Don't forget there will likely be rules you need to follow in your country although the default settings seemed to comply for me in Australia.

<https://forum.core-electronics.com.au/t/configuring-a-pico-and-lora-p2p/15879>

To install, save the above "Required Files" onto each device, renaming mainHouse.py and mainGate.py to main.py on their destination device so they execute automatically at startup.

If you don't know how to connect to your Pico and copy files there is a great example here.

<https://core-electronics.com.au/guides/how-to-setup-a-raspberry-pi-pico-and-code-with-thonny/>

To make the program code easier to modify and maintain I have decided to send the text messages from one device to the other using the standard json format. This forces things to be done in a standard way and multiple values can easily be encoded/decoded into a single text string.

The simple json format I have used looks like this.  
{Name:Value,Name:Value}

Importable built-in micropython modules exist for the Json format so it makes sense to do it this way.

## Startup

The first thing that mainGate.py does on startup is to send a message to mainHouse.py. This includes the Date and Time the gate Pico has stored. As this device had no battery backed time source this date time will be incorrect at power up. (It's something like 01 Jan 2020). Every message sent between the two devices contains the sending device's timeStamp. After this it waits for incoming messages and gate open events. Currently the only messages the gate unit will receive are time updates.

When mainHouse.py starts up it connects to the local wifi access point and updates its internal clock from a NTP server on the internet using a standard protocol Simple Network Time Protocol (SNTP). This process is performed by a library named ntpClientTZ.py. This module was based on an example found here.

<https://gist.github.com/aallan/581ecf4dc92cd53e3a415b7c33a1147c>

The modified library can offset the time for your local time zone but it is not daylight savings aware.

When mainHouse.py receives a message from mainGate.py it checks the time stamp received in the message against its own system time and if the time is out by more than one minute a message is sent back to mainGate.py with the field "TimeUpdate:True" instructing the device to update it's time with the time stamp contained in the message.

The gate device is solar powered with a battery backup. So a warning can be displayed at the house device if the voltage drops below a certain level. The gate device's voltage is contained within every message sent to the house device. A warning state can be signaled on the house device LED. The minimum voltage can be specified in constants.py.

Not that it's a big deal but anyone within range of the LoRa modules could listen in on my gate open and close events without much trouble so I decided to include encryption for the json messages strings. I found an example implementation of the inbuilt micropython ucryptolib here.

<https://github.com/iyassou/mpyaes>

In future I could think of other things to monitor and encryption as standard could be a good thing to have.

# Things to do (mainGate.py)

# =====

```
# Flash LED on gate if voltage is low.  
# Publish log from mini web server on house device  
# =====
```

Getting all the code to work was a major part of the project. I usually find the easiest way to get code working is to include debugging output within the code. I have a standard library to assist with this named debug.py. I usually leave the debugging code in place for the next guy and to assist in solving unforeseen problems and to understand how the code works. It can simply be disabled by setting the debug level to zero in constants.py. Debug output can be directed to the console and or a file.

## Error Messages

Flash the LED to Indicate an error Long flash then Short flashe(s).

1 long, 1 short = Low voltage at mainGate.py (Less than value set in constants.py)

1 long, 2 short = Unable to connect to Wifi at mainHouse.py

1 long, 3 short = Unable to set system clock at mainHouse.py

1 long, 4 short = Unable to open socket for inbound web page connection

1 long, 5 short = Heartbeat message timed out

1 long, 6 short = Unable to contact NTP time server

## How Debug Logging Works

Each call to debug passes the following values

Field Number	Description
1	Debug level that triggered the message
2	Sequential counter for the debug log
3	Name of the function that the debug message comes from
4	The actual debug message
5	A boolean to enable or disable logging the message to a file debugLog.csv

# LED Message and Error Indicator

## File Name: blink.py

The House Device has an LED connected to PIN 7(GPIO 5) and ground on pin 18

Several different status indications are sent as follows.

A “Gate Open” event adds to a counter of short flashes indicating how many times the gate has been opened since the last reset.

Error messages start with a long flash then a number of short flashes. Normally error messages are fatal and require intervention.

## Example Code

Here is some example code from blink.py so you can assess the readability of my coding style. Full code is available in the github repository.

<https://github.com/pollardd/Lora-Gate-Alarm/>

These functions blink the LED and look for a press of the button. If these two things aren't combined while the LED is flashing the button pressed could be missed. The button is used to reset the gate open count indication.

```
from machine import Pin
import time
import constants
import counters
import debug

# Constants
DEBUG = constants.DEBUG
LOGTOFILE = constants.LOGTOFILE
ENCRYPTION = constants.ENCRYPTION

button = Pin(14, Pin.IN, Pin.PULL_UP) # Physical Pin 19 Gnd = 13

def flash(ledPin, long, short):
    if(DEBUG >=2):
        debug.debug(DEBUG, "blink.flash()", " ", LOGTOFILE)

    led = Pin(ledPin, Pin.OUT)

    # Long
    count=0
    while(count < long):
        led.on()
        #print("LED ON")
```

```

        time.sleep(1.2)
        led.off()
        #print("LED OFF")
        time.sleep(0.75)
        count = count+1
# Short
count=0
while(count < short):
    led.on()
    # time.sleep(0.5)
    checkButtonPress(.3) # Check the button while holding the flash on
    led.off()
    # time.sleep(0.5)
    checkButtonPress(.3) # Check the button while holding the flash off
    count = count+1

#time.sleep(1.5)
checkButtonPress(1.25) # Check the button while waiting between flash
groups

def checkButtonPress(seconds):
    # If Button pressed clear the flash count
    # This function also acts as a timer for the main program loop
    if(DEBUG >=2):
        debug.debug(DEBUG, "blink.checkButtonPress()", " ", LOGTOFILE)

    while seconds>0:
        if(DEBUG >=3):
            debug.debug(DEBUG, "checkButtonPress(seconds)",
"seconds="+str(seconds), LOGTOFILE)

            if(getButton()==1):
                if(DEBUG >=1):
                    debug.debug(DEBUG, "checkButtonPress(seconds)      ", "Reset
Count Button Pressed, openCount=0", LOGTOFILE)
                    counters.openCount=0
                    time.sleep(0.125)
                    seconds=seconds - 0.125

def getButton():
    # Invert button value so 1=button pressed and 0=not pressed
    if(DEBUG >=2):
        debug.debug(DEBUG, "blink.getButton()", " ", LOGTOFILE)
    buttonPressed= not button.value()
    if(DEBUG >=3):
        debug.debug(DEBUG, "getButton()      ", "buttonPressed="+
str(buttonPressed), LOGTOFILE)
    return buttonPressed

```



## Message Format

Messages are sent between devices in standard text format to keep things consistent and make adding new features easier. I have come up with the following fields to be encoded into json format which are then encrypted.

Field Name	Data Description
Message Number	Sequential number generated by the sending device. House Device starts at 1 and the Gate Device starts at 10,000
TimeStamp	Time on the clock of the sending device
SrcDevice	Name of the sending device
DstDevice	Name of the target device
TimeUpdate	Should the destination device update its clock to this time? (True or False)
GateOpen	Is the gate currently open? (True or False)
Voltage	Battery Voltage of the sending device as a percentage 100% being full.
Text	Free Form Text. This can simply be a human readable description or could also be used to implement additional functions.

### Example Json Message:

```
{
  "MessageNumber": "1",
  "SrcDevice": "MainGate",
  "DstDevice": "MainHouse",
  "TimeStamp": "(2023, 1, 14, 13, 49, 34, 5, 14)",
  "TimeUpdate": "False",
  "BatteryVoltage": "99.1443",
  "GateOpen": "False",
  "TextMessage": "StartUp Message"
}
```

Here is a picture taken from the house where the House device is positioned.  
You can see where I have marked the front gate approximately 350m away.

