MacroPad 2FA TOTP Authentication Friend

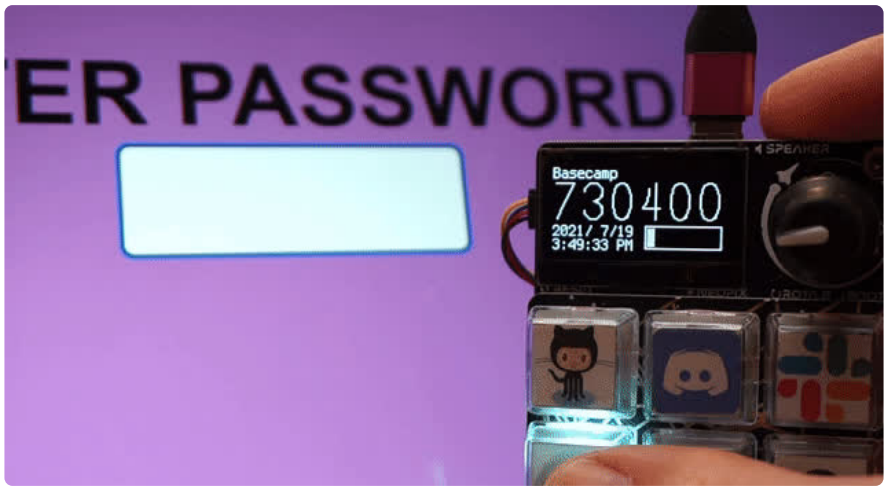
Created by Carter Nelson

https://learn.adafruit.com/macropad-2fa-totp-authentication-friend

Last updated on 2024-03-08 03:59:00 PM EST

Table of Contents

Overview



What is TOTP?

Having 2 Factor Authentication (**2FA**) on all your accounts is a good way to keep your data more secure. With 2FA logins, not only is a username and password needed, but also a one-time-use code. There are a few different ways to get that code, such as by email, phone or SMS. But my favorite way is to do it is via a 'Google Authenticator' **T**ime-based **O**ne **T**ime **P**assword, also known as a **TOTP**.

Using an app on your phone like Authy (https://adafru.it/lbW) or Authenticator, you set up a secret code given to you by the service, then every 30 seconds, a new code is generated for you. What's extra nice is that the Google Authenticator protocol is supported by just about every service and phone/tablet.

Perfect for when you don't have your phone handy, or don't want to noodle with Authy all day. Plus it types it in for you!

Why Use a MacroPad?

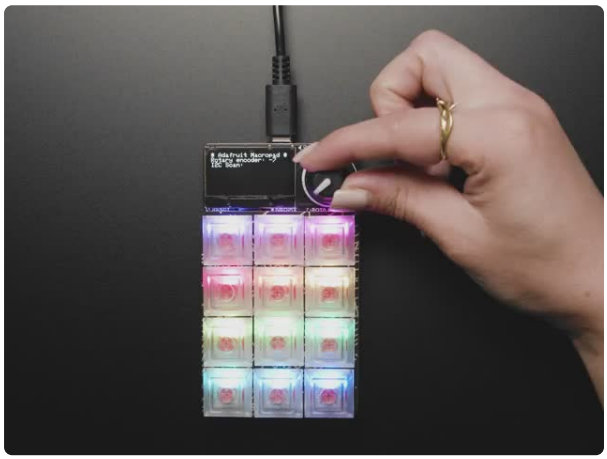
So a simple smartphone app could take care of all your TOTP needs. Why use a MacroPad? Maybe you don't have a smartphone, or don't have yours always handy near your PC. Also, by emulating a HID keyboard, the MacroPad can type in the code for you. So by having a MacroPad tethered to your PC via a USB cable, you'll have a TOTP device always at the ready.

There are also a few other guides that provide similar capabilities using different hardware platforms. Be sure to check them out as well:

- PyPortal 2FA TOTP Authentication Friend (https://adafru.it/TSD) - PyPortal based
- CircuitPython 2FA TOTP Authentication Friend (https://adafru.it/IbX) - Feather ESP8266 based

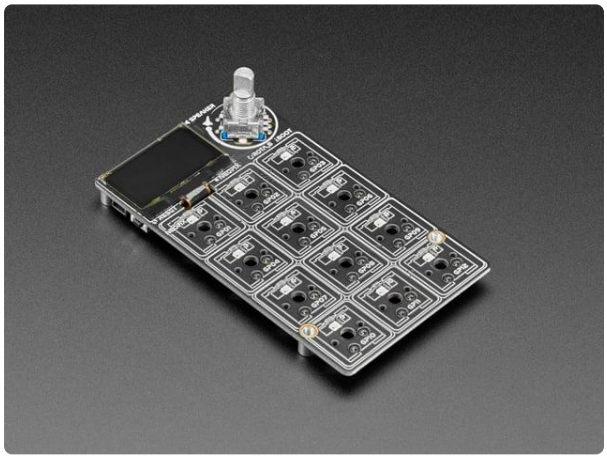
Parts

The main item needed is an Adafruit MacroPad and associated parts. You can get all of these via a kit:

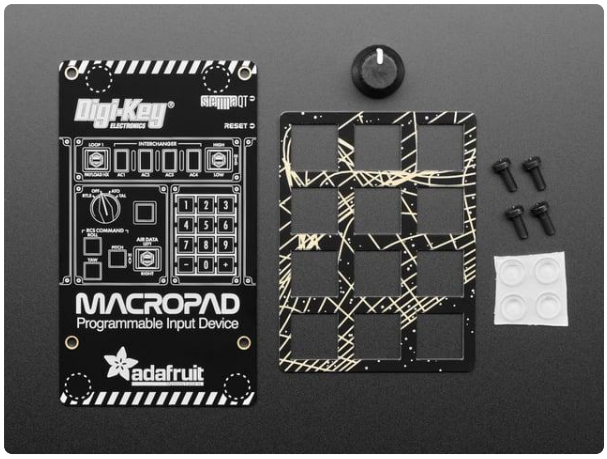


Adafruit MacroPad RP2040 Starter Kit - 3x4 Keys + Encoder + OLED
Strap yourself in, we're launching in T-minus 10 seconds...Destination? A new Class M planet called MACROPAD! M here stands for Microcontroller because this 3x4 keyboard controller...
https://www.adafruit.com/product/5128

Or a la carte if you already have some of the items or what to customize things, like use different keebs.



Adafruit MACROPAD RP2040 Bare Bones - 3x4 Keys + Encoder + OLED
Strap yourself in, we're launching in T-minus 10 seconds...Destination? A new Class M planet called MACROPAD! M here, stands for Microcontroller because this 3x4 keyboard...
https://www.adafruit.com/product/5100



Adafruit MacroPad RP2040 Enclosure + Hardware Add-on Pack

Dress up your Adafruit Macropad with PaintYourDragon's fabulous decorative silkscreen enclosure and hardware kit. You get the two custom PCBs that are cut to act as a protective...

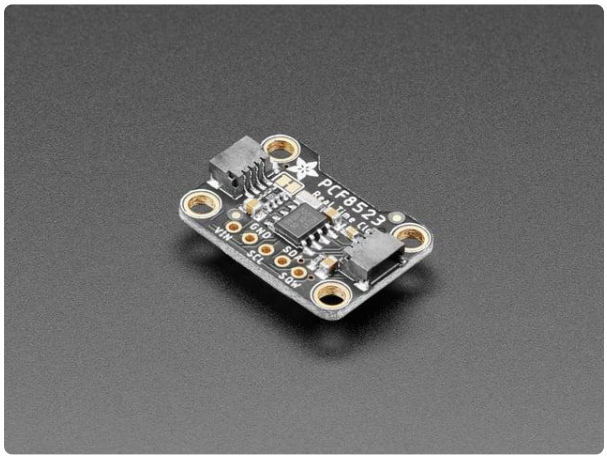
https://www.adafruit.com/product/5103

And, **if you go a la carte**, don't forget keyswitches (https://adafru.it/TSE) and keycaps (https://adafru.it/TSF).

A real-time clock (RTC) breakout is also needed to keep track of current time. Here are some options.

Use STEMMA QT Breakout

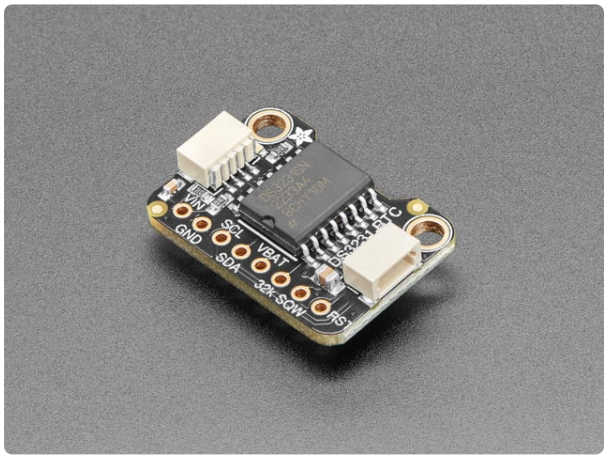
Easiest is to use a STEMMA QT capable breakout:



Adafruit PCF8523 Real Time Clock Breakout Board

This is a great battery-backed real time clock (RTC) that allows your microcontroller project to keep track of time even if it is reprogrammed, or if the power is lost. Perfect for...

https://www.adafruit.com/product/5189

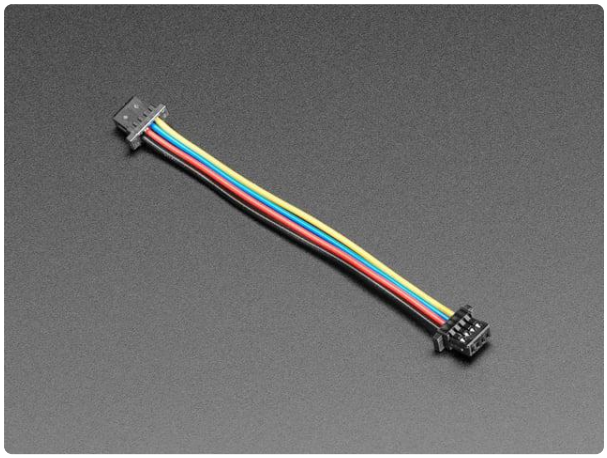


Adafruit DS3231 Precision RTC - STEMMA QT

The datasheet for the DS3231 explains that this part is an "Extremely Accurate I²C-Integrated RTC/TCXO/Crystal". And, hey, it does exactly...

https://www.adafruit.com/product/5188

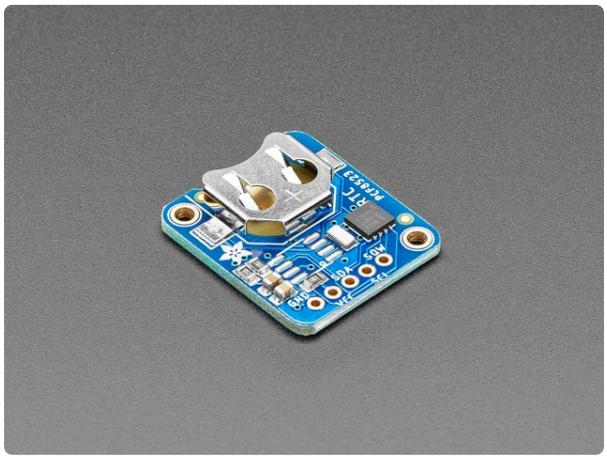
This will allow direct connection **without soldering** to the MacroPad using this STEMMA QT cable:



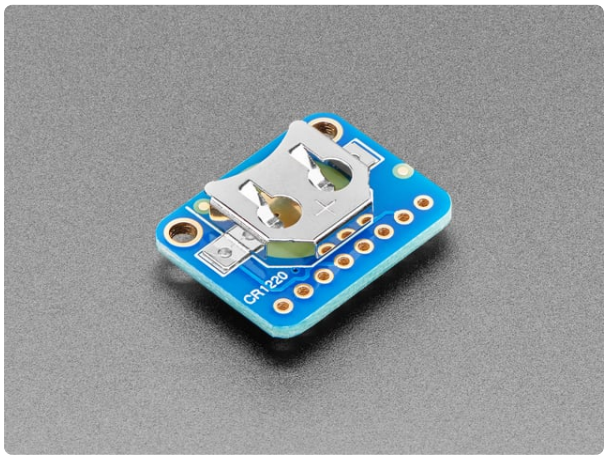
[STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long]
This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...
https://www.adafruit.com/product/4399

Use Older Style Breakout

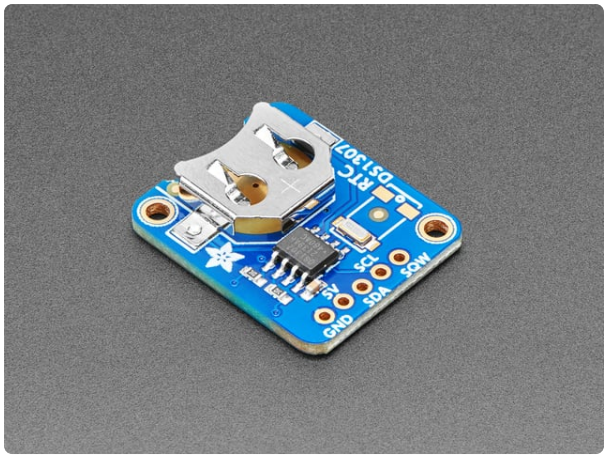
These options also work fine. However, some soldering will be required on the headers pins.



[Adafruit PCF8523 Real Time Clock Assembled Breakout Board]
This is a great battery-backed real time clock (RTC) that allows your microcontroller project to keep track of time even if it is reprogrammed, or if the power is lost. Perfect for...
https://www.adafruit.com/product/3295



[Adafruit DS3231 Precision RTC Breakout]
The datasheet for the DS3231 explains that this part is an "Extremely Accurate I^{2}C-Integrated RTC/TCXO/Crystal". And, hey, it does exactly what it says...
https://www.adafruit.com/product/3013

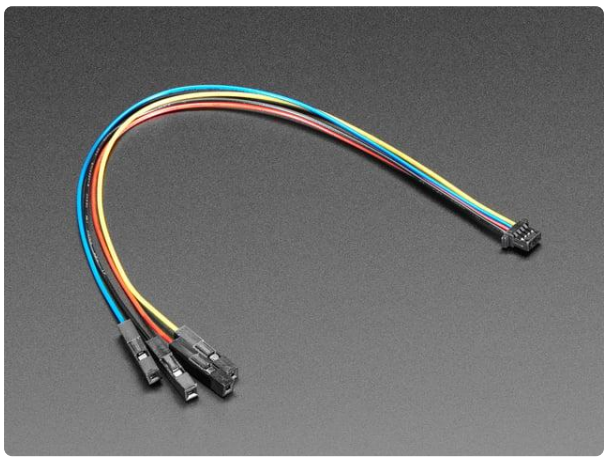


Adafruit DS1307 Real Time Clock Assembled Breakout Board

This is a great battery-backed real time clock (RTC) that allows your microcontroller project to keep track of time even if it is reprogrammed, or if the power is lost. Perfect for...

https://www.adafruit.com/product/3296

With male headers soldered on the RTC breakouts, this STEMMA QT cable can be used for connecting to the MacroPad:



STEMMA QT / Qwiic JST SH 4-pin Cable with Premium Female Sockets

This 4-wire cable is a little over 150mm / 6" long and fitted with JST-SH female 4-pin connectors on one end and premium female headers on the other. Compared with the chunkier...

https://www.adafruit.com/product/4397

Batteries Not Included

These RTC breakouts all use the same CR1220 coin cell battery:

CR1220 12mm Diameter - 3V Lithium Coin Cell Battery

These are the highest quality & capacity batteries, the same as shipped with the iCufflinks, iNecklace, Datalogging and GPS Shields, GPS HAT, etc. One battery per order...

https://www.adafruit.com/product/380

FAQ

THIS IS NOT A QUESTION MORE OF A COMMENT. YOU ARE PROGRAMMING THE TOTP SECRET INTO THE FLASH OF THE MICROCONTROLLER AND ITS NOT ENCRYPTED OR PROTECTED AT ALL ANYONE COULD BREAK INTO YOUR APARTMENT, GO TO YOUR BEDROOM, LOOK ON YOUR DESK, FIND THIS AND THEN CONNECT IT UP TO THEIR HACKER LAPTOP TO GRAB YOUR SECRET KEY THEN IF THEY HAD YOUR USERNAME AND PASSWORD THEY WOULD BE ABLE TO LOG IN AS YOU AND THIS IS REALLY INSECURE ITS SO IRRESPONSIBLE TO CONSIDER PUBLISHING A PROJECT LIKE THIS BY THE WAY DID YOU SEE THAT SNOWDEN APP? MAYBE YOU CAN RUN THAT ON A PHONE SO YOU CAN WATCH YOUR DESK REMOTELY AND MAKE SURE NOBODY BROKE IN TO STEAL YOUR MACROPAD? OH WAIT YOU JUST SAID YOU DON'T HAVE A PHONE. OK I DONT KNOW WHAT MY QUESTION IS

this project is probably not for you

CircuitPython

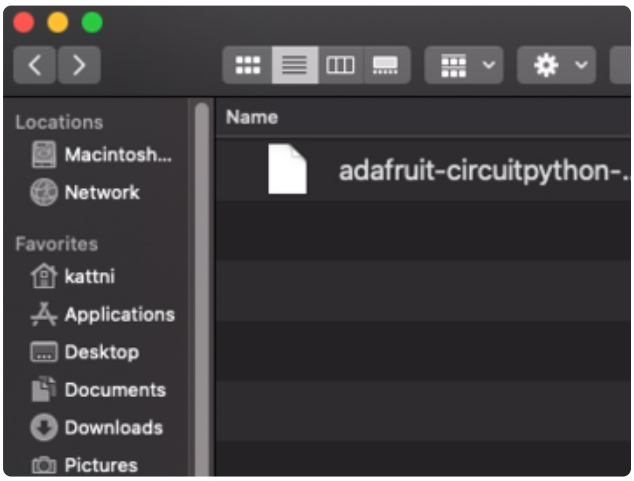
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

CircuitPython Quickstart

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

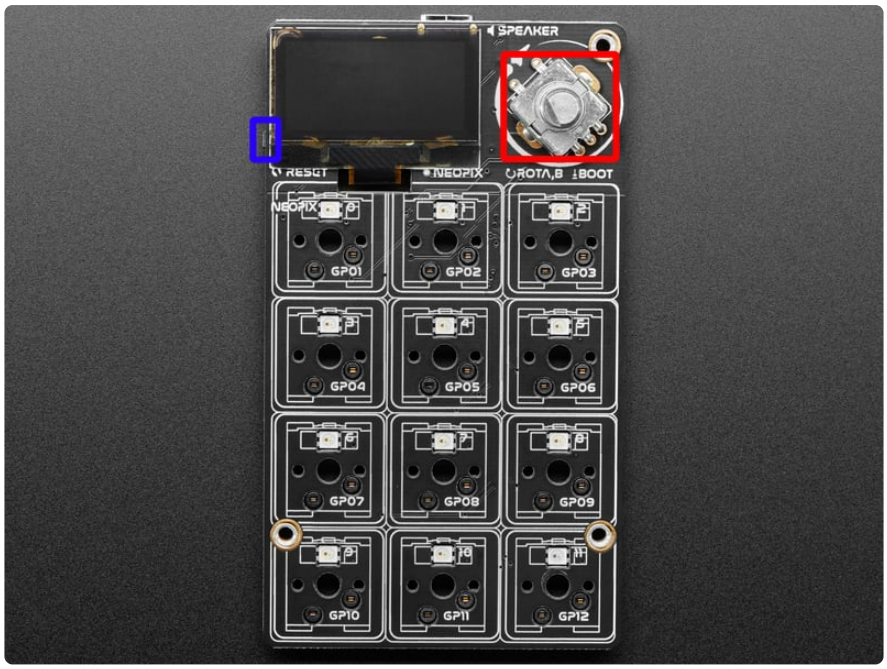
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/TB9



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

Save it wherever is convenient for you.



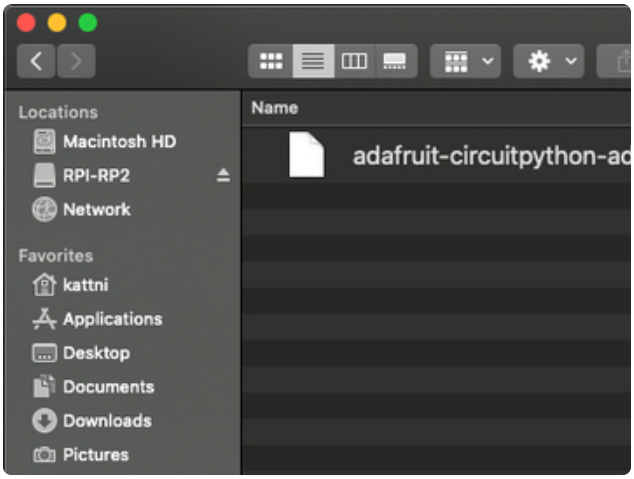
The BOOT button is the button switch in the rotary encoder! To engage the BOOT button, simply press down on the rotary encoder.

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!**

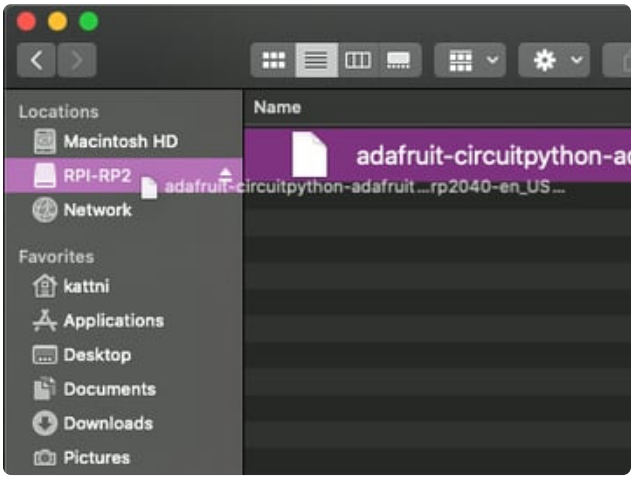
If the drive does not appear, release all the buttons, and then repeat the process above.

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.

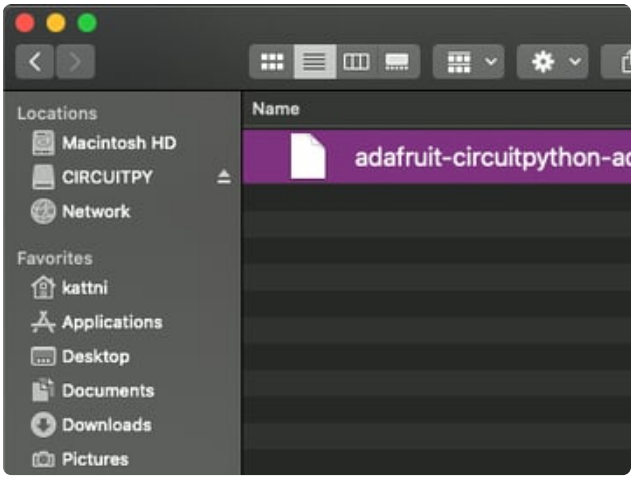
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



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 doesn't even show up as a disk drive when installing CircuitPython, try loading this 'nuke' UF2 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

Project Code

Your project will use a specific set of CircuitPython libraries and the **code.py** file, along with a folder full of key configuration files. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Drag the contents of the uncompressed bundle directory onto your MACROPAD board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

```
# SPDX-FileCopyrightText: 2021 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
# base hardware stuff
import board
import rtc
import keypad
import rotaryio
import neopixel
# crypto stuff
import adafruit_pcf8523
import adafruit_hashlib as hashlib
# UI stuff
import displayio
import terminalio
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text import label
from adafruit_progressbar.horizontalprogressbar import HorizontalProgressBar
# HID keyboard stuff
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS
from adafruit_hid.keycode import Keycode

#--| User Config |-------------------------------------------------------
UTC_OFFSET = -4        # time zone offset
USE_12HR = True        # set 12/24 hour format
DISPLAY_TIMEOUT = 60   # screen saver timeout in seconds
DISPLAY_RATE = 1       # screen refresh rate
#-----------------------------------------------------------------------

# Get secrets from a secrets.py file
try:
    from secrets import secrets
    totp_keys = secrets["totp_keys"]
except ImportError:
    print("Secrets are kept in secrets.py, please add them there!")
    raise
```

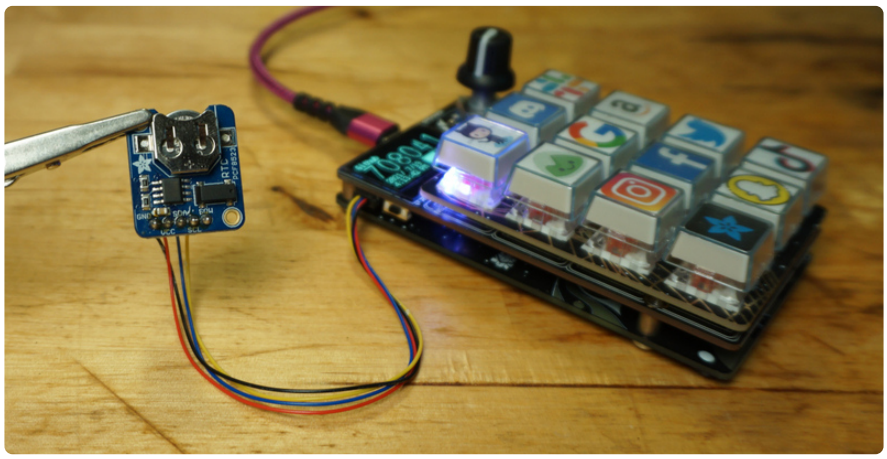
```
except KeyError:
    print("TOTP info not found in secrets.py.")
    raise

# set board to use PCF8523 as its RTC
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
pcf = adafruit_pcf8523.PCF8523(i2c)
rtc.set_time_source(pcf)

#-------------------------------------------------------------------------
#                       H I D    S E T U P
#-------------------------------------------------------------------------
time.sleep(1)  # Sleep for a bit to avoid a race condition on some systems
keyboard = Keyboard(usb_hid.devices)
keyboard_layout = KeyboardLayoutUS(keyboard)  # We're in the US :)

#-------------------------------------------------------------------------
#                     D I S P L A Y    S E T U P
#-------------------------------------------------------------------------
display = board.DISPLAY

# Secret Code font by Matthew Welch
# http://www.squaregear.net/fonts/
font = bitmap_font.load_font("/secrcode_28.bdf")

name = label.Label(terminalio.FONT, text="?"*18, color=0xFFFFFF)
name.anchor_point = (0.0, 0.0)
name.anchored_position = (0, 0)

code = label.Label(font, text="123456", color=0xFFFFFF)
code.anchor_point = (0.5, 0.0)
code.anchored_position = (display.width // 2, 15)

rtc_date = label.Label(terminalio.FONT, text="2021/01/01")
rtc_date.anchor_point = (0.0, 0.5)
rtc_date.anchored_position = (0, 49)

rtc_time = label.Label(terminalio.FONT, text="12:34:56 AM")
rtc_time.anchor_point = (0.0, 0.5)
rtc_time.anchored_position = (0, 59)

progress_bar = HorizontalProgressBar((68, 46), (55, 17), bar_color=0xFFFFFF,
min_value=0, max_value=30)

splash = displayio.Group()
splash.append(name)
splash.append(code)
splash.append(rtc_date)
splash.append(rtc_time)
splash.append(progress_bar)

display.root_group = splash

#-------------------------------------------------------------------------
#                     H E L P E R    F U N C S
#-------------------------------------------------------------------------
def timebase(timetime):
    return (timetime - (UTC_OFFSET*3600)) // 30

def compute_codes(timestamp):
    codes = []
    for key in totp_keys:
        if key:
            codes.append(generate_otp(timestamp, key[1]))
        else:
            codes.append(None)
    return codes
```

```
def HMAC(k, m):
    """# HMAC implementation, as hashlib/hmac wouldn't fit
    From https://en.wikipedia.org/wiki/Hash-based_message_authentication_code

    """
    SHA1_BLOCK_SIZE = 64
    KEY_BLOCK = k + (b'\0' * (SHA1_BLOCK_SIZE - len(k)))
    KEY_INNER = bytes((x ^ 0x36) for x in KEY_BLOCK)
    KEY_OUTER = bytes((x ^ 0x5C) for x in KEY_BLOCK)
    inner_message = KEY_INNER + m
    outer_message = KEY_OUTER + hashlib.sha1(inner_message).digest()
    return hashlib.sha1(outer_message)

def base32_decode(encoded):
    missing_padding = len(encoded) % 8
    if missing_padding != 0:
        encoded += '=' * (8 - missing_padding)
    encoded = encoded.upper()
    chunks = [encoded[i:i + 8] for i in range(0, len(encoded), 8)]

    out = []
    for chunk in chunks:
        bits = 0
        bitbuff = 0
        for c in chunk:
            if 'A' <= c <= 'Z':
                n = ord(c) - ord('A')
            elif '2' <= c <= '7':
                n = ord(c) - ord('2') + 26
            elif c == '=':
                continue
            else:
                raise ValueError("Not base32")
            # 5 bits per 8 chars of base32
            bits += 5
            # shift down and add the current value
            bitbuff <<= 5
            bitbuff |= n
            # great! we have enough to extract a byte
            if bits >= 8:
                bits -= 8
                byte = bitbuff >> bits  # grab top 8 bits
                bitbuff &= ~(0xFF << bits)  # and clear them
                out.append(byte)  # store what we got
    return out

def int_to_bytestring(int_val, padding=8):
    result = []
    while int_val != 0:
        result.insert(0, int_val & 0xFF)
        int_val >>= 8
    result = [0] * (padding - len(result)) + result
    return bytes(result)

def generate_otp(int_input, secret_key, digits=6):
    """ HMAC -> OTP generator, pretty much same as
    https://github.com/pyotp/pyotp/blob/master/src/pyotp/otp.py

    """
    if int_input < 0:
        raise ValueError('input must be positive integer')
    hmac_hash = bytearray(
        HMAC(bytes(base32_decode(secret_key)),
             int_to_bytestring(int_input)).digest()
    )
    offset = hmac_hash[-1] & 0xf
    code = ((hmac_hash[offset] & 0x7f) << 24 |
            (hmac_hash[offset + 1] & 0xff) << 16 |
```

```
            (hmac_hash[offset + 2] & 0xff) << 8 |
            (hmac_hash[offset + 3] & 0xff))
    str_code = str(code % 10 ** digits)
    while len(str_code) < digits:
        str_code = '0' + str_code

    return str_code

#---------------------------------------------------------------------
#                   M A C R O P A D    S E T U P
#---------------------------------------------------------------------
key_pins = (
    board.KEY1,
    board.KEY2,
    board.KEY3,
    board.KEY4,
    board.KEY5,
    board.KEY6,
    board.KEY7,
    board.KEY8,
    board.KEY9,
    board.KEY10,
    board.KEY11,
    board.KEY12,
    board.BUTTON,
)

keys = keypad.Keys(key_pins, value_when_pressed=False, pull=True)

knob = rotaryio.IncrementalEncoder(board.ROTA, board.ROTB)

pixels = neopixel.NeoPixel(board.NEOPIXEL, 12)
pixels.fill(0)

#####################################
# MAIN
#####################################
awake = True
knob_pos = knob.position
current_key = key_pressed = 0
last_compute = last_update = wake_up_time = time.time()
totp_codes = compute_codes(timebase(last_compute))
while True:
    now = time.time()
    progress_bar.value = now % 30
    event = keys.events.get()
    # wakeup if knob turned or button pressed
    if knob.position != knob_pos or event:
        if not awake:
            last_update = 0 # force an update
        awake = True
        knob_pos = knob.position
        wake_up_time = now
    # handle key presses
    if event:
        if event.pressed:
            key_pressed = event.key_number
            # knob
            if key_pressed == 12:
                keyboard_layout.write(totp_codes[current_key])
                keyboard.send(Keycode.ENTER)
            # keeb
            elif key_pressed != current_key:
                # is it a configured key?
                if totp_keys[key_pressed]:
                    current_key = key_pressed
                    pixels.fill(0)
                    last_update = 0 # force an update
    # update codes
```

```
    if progress_bar.value < 0.5 and now - last_compute > 2:
        totp_codes = compute_codes(timebase(now))
        last_compute = now
    # update display
    if now - last_update > DISPLAY_RATE and awake:
        pixels[current_key] = totp_keys[current_key][2]
        name.text = totp_keys[current_key][0][:18]
        code.text = totp_codes[current_key]
        tt = time.localtime()
        if USE_12HR:
            hour = tt.tm_hour % 12
            ampm = "AM" if tt.tm_hour < 12 else "PM"
        else:
            hour = tt.tm_hour
            ampm = ""
        rtc_date.text = "{:4}/{:2}/{:2}".format(tt.tm_year, tt.tm_mon, tt.tm_mday)
        rtc_time.text = "{}:{:02}:{:02} {}".format(hour, tt.tm_min, tt.tm_sec, ampm)
        last_update = now
        splash.hidden = False
    # go to sleep after inactivity
    if awake and now - wake_up_time > DISPLAY_TIMEOUT:
        awake = False
        knob_pos = knob.position
        pixels.fill(0)
        splash.hidden = True
```

RTC Setup



As the name implies, **Time**-based One Time Passwords rely on **time**. For any internet connected device, this can be done fairly easy using a NTP service. However, the MacroPad does not have any wifi capabilities, so can not readily connect to the internet. To get around this, we will use an external time source, a Real Time Clock (**RTC**) device, directly connected to the MacroPad.

An RTC has a battery backup to maintain time tracking even when the rest of the system is powered down.

Connecting RTC

In this guide we use a PCF8523 RTC breakout (http://adafru.it/3295). But you can use any other I2C RTC breakout, as long as it has a CircuitPython driver library. Follow the assembly instructions in the PCF8523 guide (https://adafru.it/TTa) to solder on the header pins.

Once the header pins are soldered on, use the STEMMA QT cable (http://adafru.it/4397) and connect:

- **PCF8523 GND** to **STEMMA QT black** cable
- **PCF8523 VCC** to **STEMMA QT red** cable
- **PCF8523 SDA** to **STEMMA QT blue** cable
- **PCF8523 SCL** to **STEMMA QT yellow** cable

The SQW pin does not need to be connected.

Setting the RTC

Use the **rtc_setter.py** code below and edit the time values to be something many seconds into the future. Edit these lines:

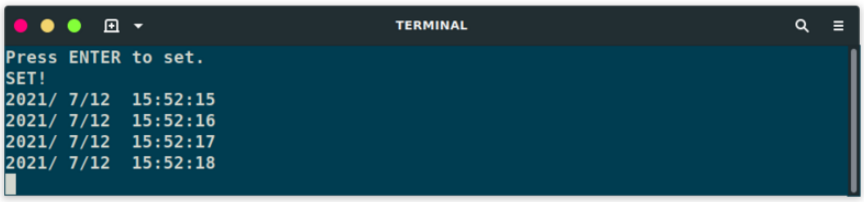
```
# values to set
YEAR = 2021
MON = 1
DAY = 1
HOUR = 12
MIN = 23
SEC = 42
```

Copy the **rtc_setter.py** code to your **CIRCUITPY** folder. Then, connect to the CircuitPython REPL and run the code as follows:

```
import rtc_setter
```

It will show the time you set and wait until you press ENTER to actually set the time.

Now watch some other time source that is showing the current time. When the time you set arrives, press ENTER.



The RTC will now be set to that time. The code will loop forever showing the RTC time as a way to check that is actually set and working. You can press <CTRL><C> to break out of the code or just reset.

Make sure you have a fresh coin cell battery in the RTC.

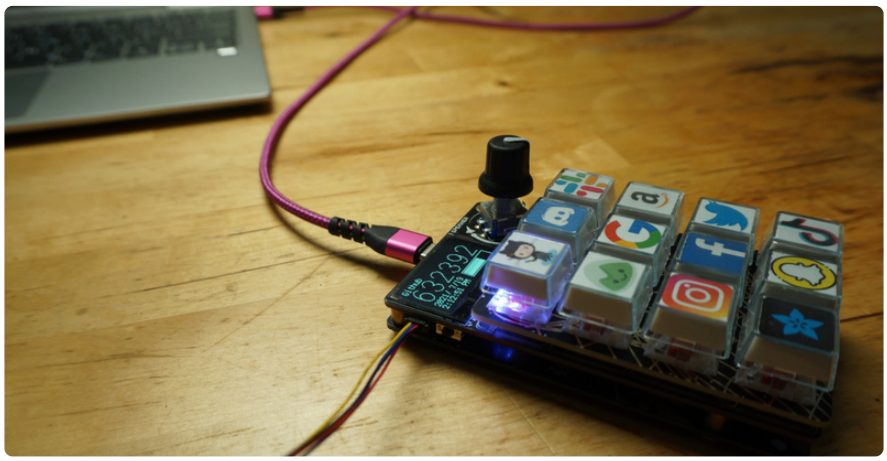
RTC Setter Code

Here is the RTC setter code **rtc_setter.py**:

```
# SPDX-FileCopyrightText: 2021 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import adafruit_pcf8523

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
pcf = adafruit_pcf8523.PCF8523(i2c)

# values to set
YEAR = 2021
MON = 1
DAY = 1
HOUR = 12
MIN = 23
SEC = 42

print("Ready to set RTC to: {:4}/{:2}/{:2}  {:2}:{:02}:{:02}".format(YEAR,
                                                                     MON,
                                                                     DAY,
                                                                     HOUR,
                                                                     MIN,
                                                                     SEC))
_ = input("Press ENTER to set.")

pcf.datetime = time.struct_time((YEAR, MON, DAY, HOUR, MIN, SEC, 0, -1, -1))

print("SET!")

while True:
    now = pcf.datetime
    print("{:4}/{:2}/{:2}  {:2}:{:02}:{:02}".format(now.tm_year,
                                                    now.tm_mon,
```

```
                                                now.tm_mday,
                                                now.tm_hour,
                                                now.tm_min,
                                                now.tm_sec))
    time.sleep(1)
```

Code Usage



Sanity Check

The **secrets.py** file included with the project bundle zip contains some predefined fake codes. These aren't actual secrets, so nothing is being given away here. The following site can be used to check that everything is working:

TOTP Token Generator

https://adafru.it/TTb

The first key on the MacroPad is setup with the site's default secret key of `JBSWY3DPEHPK3PXP`. So if your RTC is properly setup, then the same resulting 6 digit code should appear on both your MacroPad and the website. If they don't agree, double check that your RTC is synced well with UTC time.

World Clock

https://adafru.it/TTc

Go back and re-run the RTC setter code if needed.

Also make sure you've adjusted this line of code:

```
UTC_OFFSET = -4        # time zone offset
```

to match your local time zone's offset relative to UTC. Good info and global map over on wikipedia:

UTC offsets

https://adafru.it/ifS

Also take into account if daylight saving is active since the MacroPad can not determine that automatically. Do this by simply adjusting the `UTC_OFFSET` value by one hour as needed.

Configuring Keys

To configure the keys, you edit the the **secrets.py** file in your **CIRCUITPY** folder. The example **secrets.py** in the project bundle zip you downloaded earlier can be used as guide. Inside, you'll see something like this:

```
secrets = {
                  # tuples of name, sekret key, color
    'totp_keys' : [("Github", "JBSWY3DPEHPK3PXP", 0x8732A8),
                   ("Discord", "JBSWY3DPEHPK3PXQ", 0x32A89E),
                   ("Slack", "JBSWY5DZEHPK3PXR", 0xFC861E),
                   ("Basecamp", "JBSWY6DZEHPK3PXS", 0x55C24C),
                   ("Gmail", "JBSWY7DZEHPK3PXT", 0x3029FF),
                   None,
                   None, # must have 12 entires
                   None, # set None for unused keys
                   None,
                   ("Hello Kitty", "JBSWY7DZEHPK3PXU", 0xED164F),
                   None,
                   None,
                  ]
    }
```

The main `secrets` parameter is a Python dictionary. In this example, it only has one entry - the `totp_keys` entry. This is a list with 12 entries, one per key. **You must keep this list length to 12.** Each entry is a 3 tuple:

```
(name, code, color)
```

with the following contents:

- **name** - A string name for the site the key is associated with.
- **code** - The secret code the site provided when you set up 2FA.
- **color** - The NeoPixel color to use when the key is active.

Here is an example:

```
("Github", "JBSWY3DPEHPK3PXP", 0x8732A8)
```

When editing this file, be careful not to accidentally delete any of the square brackets, parans, or commas. Otherwise you'll end up with a syntax error when the code tries to import the secrets.

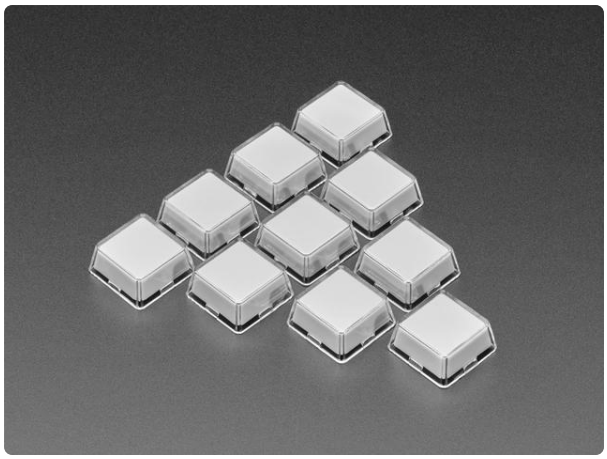
Selecting and Sending Code

Selecting codes is easy, just press the key you assigned the code to. The OLED will update to show the code for that site and the NeoPixel under the key will light up the color you set.

If you want the MacroPad to send the keystrokes for the code, as if you typed them on a keyboard, press the knob button down.

Customizable Keys

If you want to have site specific icons for each of the keys, these relegendable key caps are handy:



Relegendable Plastic Keycaps for MX Compatible Switches 10 pack
Get ready to customize your keeb with a 10 pack of two-part plastic keycaps for your next mechanical keyboard or https://www.adafruit.com/product/5039

But these are not required. The site name is shown on the OLED. So that can be used to determine which site's pass code is currently being shown.

To make custom icons, print each out sized to a 17/32" x 17/32" square and cut them out. Removing the clear tops on the relegendable keys is best done with the keycaps mounted on a key switch. Then, just place the printed cutout on the keycap and put the plastic cover back on.

Here are some PDFs you can use to help. One is blank and the second includes some example icons that were used for this guide.

keycaps_blank.pdf

https://adafru.it/U4b

keycaps_example_icons.pdf

https://adafru.it/U4c