Table of Contents

**Overview**
- Parts 4

**Macropad Assembly**
- Switches into Plate 6
- Connect to Board 7
- Add Switches 7
- Backplate 8
- Keycaps 9

**Lock Circuit**
- Optional Solenoid Lock Driver Circuit 10
- Power Supply Requirements 11
- For high power/current needs 12

**CircuitPython**
- CircuitPython Quickstart 12
- Safe Mode 14
- Flash Resetting UF2 16

**Code**
- Text Editor 17
- Download the Project Bundle 17
- Passcode Changes and Security 20
- Files 21

**Use**
- START Mode 22
- ENTRY Mode 23
- OPEN Mode 23
- FAIL Mode 23
- Pressing START during ENTRY 23
Overview

Electronic keypads are often used to enter a password to perform a function, such as open a door in a security application. No key is needed, only memorization of a passcode.

Many keypad implementations suffer from several security issues which make a keypad entry system much less secure. Using the same keys on the keypad over and over may lead to someone in the vicinity seeing the pattern of keystrokes used. Also, over time the used keys will start to wear down, allowing a person to more easily guess the digits used on the keypad.

There are methods to thwart the shortcomings noted above. One such was implemented by Stephen B. Hirsch circa 1980. He developed a lock which scrambled where the digits would be located on the keypad. This prevented the use of pattern memorization and created even wear on the keys. There is a demonstration video on YouTube (https://adafru.it/UBZ).

The Hirsch Scramblepad (https://adafru.it/Egy) required 7-segment displays on the keys, a costly proposition. But that can be simulated on the Adafruit MacroPad.

This project will use the much, much less expensive (and much more fun) Adafruit MacroPad to simulate some of the high security features of locks such as the Hirsch Scramblepad. As the keys do not have displays in them (only colorful LEDs underneath), the scrambled numbers will be displayed on the MacroPad display.
The MacroPad will include an indication of success and failure. Optionally, the digital pins on the STEMMA QT port can be used to trigger opening of a solenoid lock with a bit of standard electronics, if desired.

Parts

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...
https://www.adafruit.com/product/5128

USB Type A to Type C Cable - approx 1 meter / 3 ft long
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/4474
Optional keycaps for the lower left and right of the keypad:

**Black Windowed Lamp R4 Keycap for MX Compatible Switches**

These stark and elegant Black Windowed Lamp R4 Keycaps are made of opaque black plastic with a small cutout in the south-center edge that wraps around the top and...

[https://www.adafruit.com/product/5112](https://www.adafruit.com/product/5112)

To make the optional solenoid lock driver:

**Lock-style Solenoid - 12VDC**

Solenoids are basically electromagnets: they are made of a big coil of copper wire with an armature (a slug of metal) in the middle. When the coil is energized, the slug is pulled into...

[https://www.adafruit.com/product/1512](https://www.adafruit.com/product/1512)

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Description</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIP120 Power Darlington Transistor</td>
<td>1</td>
<td></td>
<td><a href="https://www.adafruit.com/product/976">https://www.adafruit.com/product/976</a></td>
</tr>
<tr>
<td>Breadboard Half-size</td>
<td>1</td>
<td></td>
<td><a href="https://www.adafruit.com/product/64">https://www.adafruit.com/product/64</a></td>
</tr>
<tr>
<td>1N4001 Diode 10 Pack (1 used)</td>
<td>1</td>
<td></td>
<td><a href="https://www.adafruit.com/product/755">https://www.adafruit.com/product/755</a></td>
</tr>
<tr>
<td>Female DC Power adapter</td>
<td>1</td>
<td>- 2.1mm jack to screw terminal block</td>
<td><a href="https://www.adafruit.com/product/368">https://www.adafruit.com/product/368</a></td>
</tr>
<tr>
<td>STEMMA QT / Qwiic JST SH Cable</td>
<td>1</td>
<td>4-pin to Male Headers Cable - 150mm Long</td>
<td><a href="https://www.adafruit.com/product/4209">https://www.adafruit.com/product/4209</a></td>
</tr>
</tbody>
</table>
Macropad Assembly

The Macropad features hot-swap sockets for the switches -- gone are the days of having to commit to one type of switch and solder it down! Now, you can plug in your Cherry MX red keyswitches, use them for a while, get bored, decide its time to test out some lubed, filmed, re-sprung Invyr Holy Pandas, and swap them just like that!

Switches into Plate
First, insert a couple pf keyswitches through the keyswitch plate. The plate mechanically connects the switches to each other, which lends some nice lateral stability to the keys.
Connect to Board
Carefully press the two switches into the switch sockets, being very careful to align the legs so none bend!

Add Switches
Continue adding switches, being mindful of their orientation.
Backplate
You can add the optional backplate using four M3 x 6mm screws.
Keycaps
Now, you can add your keycaps! simply press them onto the keyswitch stems until they are fully seated.
Lock Circuit

When the correct code is entered into the lock, the key on the lower right will glow green and the screen will say Open. For educational use that's good. But for an action, such as opening a lock, additional hardware is required.

Optional Solenoid Lock Driver Circuit

The MacroPad has very limited digital input and output, mainly one STEMMA QT/Qwiic I2C connector on the left side of the keypad under the display. There are two signals, SDA and SCL for I2C, along with power and ground.

The optional lock activation uses the SDA pin as a digital output line, in addition to the ground connection. It's non-standard to how things are to work for a STEMMA QT connection, but this use is electrically and programmatically just fine.

The circuit to take the digital signal and translate that into powering a higher voltage, high current lock/solenoid is shown below.
The SDA and Ground (Blue and Black, respectively) from the STEMMA QT connection are on the left. The digital signal feeds a TIP120 Darlington transistor at it's base through a 2.2K ohm current limiting resistor. The emitter from the transistor is grounded and the collector is connected through the solenoid to power. When the digital signal from the MacroPad SDA goes high, it triggers the transistor like a switch, sending power through the solenoid to ground, making the solenoid retract the bolt.

A 1N4001 transistor is used across the solenoid. When the power no longer flows through the solenoid, a collapsing magnetic field will cause a reverse voltage. The reverse voltage flows through the diode and not the transistor, protecting the transistor from the dangerous spike.

Power Supply Requirements

The voltage and current in to the circuit via the jack shown should match the voltage and current requirements of the solenoid you use.

The Adafruit #1512 solenoid uses 9 to 12 volts DC and draws 650mA at 12V, 500 mA at 9V when activated (per the product page [https://adafruit.it/UB-]). I used a 9 volt, 1 amp supply as the current from the supply should be more than the maximum the
solenoid needs (to keep the power supply from reaching it's maximum often, which would lowering its lifespan). Adafruit #63 (https://adafru.it/dO6) works great.

If you decide on a 12 volt supply (which will retract the bolt a bit more forcefully), it needs 650 ma. The Adafruit #798 (https://adafru.it/eny) 12V 1A supply would do well.

For high power/current needs

For any high power switching, you should consider two solutions:

For higher voltage DC devices, consider a circuit similar to the one above but replace the solenoid with a 9-12V relay. You can connect the relay outputs to a suitable lock and power source.

For wall power (mains, 110/230 volt), a relay with suitable output handling capabilities could be used. The safer alternative is to use a Controllable Four Outlet Power Relay Module (https://adafru.it/yF2) version 2 - (Power Switch Tail Alternative). It will take the digital output from the STEMMA QT and switch several outlets (110 volt only).

Do not switch mains/wall power AC devices with wither microcontroller or transistor outputs, it will burn out your electronics and could start fires. Use a suitable isolation device like mentioned above, specifically designed to switch appliances with digital outputs.

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.

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 in CircuitPython 6.x**

To enter safe mode when using CircuitPython 6.x, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 700ms. On some boards, the onboard status LED (highlighted in green above) will turn solid yellow during this time. If you press reset during that 700ms, 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.)
Entering Safe Mode in CircuitPython 7.x

To enter safe mode when using CircuitPython 7.x, 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

Once you've entered safe mode successfully in CircuitPython 6.x, the LED will pulse yellow.

If you successfully enter safe mode on CircuitPython 7.x, 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.
Code

Text Editor

Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in this guide (https://adafruit.it/ANO).

Alternatively, you can use any text editor that saves simple text files.

Be sure you are running the latest version of CircuitPython to ensure updates and fixes are incorporated into the software.

Download the Project Bundle

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.

Plug your assembled MacroPad into your computer via a known good USB cable. The board should show up as a new flash drive in your file explorer/finder named CIRCUITPY.

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

```python
# Scramblepad - a random scramble keypad simulation for Adafruit MACROPAD.
# SPDX-License-Identifier: MIT
import time
import random
import board
from digitalio import DigitalInOut, Direction
import displayio
import terminalio
from adafruit_display_shapes.rect import Rect
from adafruit_display_text import label
from adafruit_macropad import MacroPad
```
# CONFIGURABLES ------------------------
# Password information
# For higher security, place password in a separate file like secrets.py
PASSWORD = "2468"
PASSWORD_LENGTH = len(PASSWORD)

# States keypad may be in
STATE_ENTRY = 1
STATE_CLEAR = 2
STATE_RESET = 3

# Color defines for keys
WHITE = 0xFFFFF
BLACK = 0x000000
RED = 0xFF0000
ORANGE = 0xFFFFA500
YELLOW = 0xFFFFF
GREEN = 0x00FF00
BLUE = 0x0000FF
PURPLE = 0x800080
PINK = 0xFFC0CB
TEAL = 0x2266AA
MAGENTA = 0xFF00FF
CYAN = 0x00FFFF

colors = [PINK, RED, ORANGE, YELLOW, GREEN, BLUE, PURPLE, TEAL, MAGENTA, CYAN]
current_colors = []

# Define sounds the keypad makes
tones = (440, 220, 245, 330, 440)  # Initial tones while scrambling
press_tone = 660  # This tone is used when each key is pressed

# Initial key values - this list will be scrambled by the scramble function
key_values = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Define the STEMMA QT I2C line SDA to be a digital output (nonstandard use)
# SDA will be used as a digital pin to trigger a transistor to activate a solenoid which will unlock, like a door
solenoid = DigitalInOut(board.SDA)
solenoid.direction = Direction.OUTPUT

# FUNCTIONS ------------------
def keys_clear():  # Set display in the Start mode, key LEDs off
    for i in range(12):
        macropad.pixels[i] = 0x000000
        group[i].text = "   "
    macropad.pixels.show()
    group[9].text = "START"
    macropad.display.show(group)
    macropad.display.refresh()

def scramble():  # Scramble values of the keys and display on screen
    for times in range(5):
        # The following lines implement a random.shuffle method
        # See https://www.rosettacode.org/wiki/Knuth_shuffle#Python
        # random.shuffle(key_values)           # Shuffle the key array
        for i in range(len(key_values)-1, 0, -1):
            j = random.randrange(i + 1)
            key_values[i], key_values[j] = key_values[j], key_values[i]
        keys_display()
        macropad.play_tone(tones[times], 0.3)  # Play a tone each scramble iteration
time.sleep(0.01)

def keys_display():  # Display the current values of the keys on screen
    for k in range(9):  # The first 9 keys
        group[k].text = str(key_values[k])
```python
macropad.pixels[k] = colors[key_values[k]]  # The 'Zero' position number
group[10].text = str(key_values[9])  # The 'Zero' position number
group[9].text = ""  # Start blanks
group[11].text = ""  # Status blanks
macropad.pixels[10] = colors[key_values[9]]
macropad.display.refresh()
macropad.pixels.show()

# INITIALIZATION -----------------------
macropad = MacroPad()  # Set up MacroPad library and behavior
macropad.display.auto_refresh = False
macropad.pixels.auto_write = False

# Set up displayio group with all the labels
group = displayio.Group()
for key_index in range(12):
x = key_index % 3
y = key_index // 3
group.append(label.Label(terminalio.FONT, text='', color=0xFFFFFF,
                         anchored_position=((macropad.display.width - 1) * x / 2,
                         macropad.display.height - 1 - (3 - y) * 12),
                         anchor_point=(x / 2, 1.0)))
group.append(Rect(0, 0, macropad.display.width, 12, fill=0xFFFFFF))
group.append(label.Label(terminalio.FONT, text='ScramblePad', color=0x000000,
                         anchored_position=(macropad.display.width//2, -2),
                         anchor_point=(0.5, 0.0)))

# Initialize in a clear state
state = STATE_CLEAR
macropad.keyboard.release_all()
keys_clear()
solenoid.value = False

# MAIN LOOP ----------------------------
while True:
    if state == STATE_RESET:
        print("Reset state")
        macropad.keyboard.release_all()
        password_guess = ""  # Reset password entry
        state = STATE_CLEAR  # Reset state

    # Check for key presses/releases
    event = macropad.keys.events.get()
    if not event:
        continue
    key_number = event.key_number
    pressed = event.pressed

    if pressed:
        if state == STATE_CLEAR:
            if key_number != 9:  # Waiting to hit START
                print("You must press start, lower left")
                macropad.keyboard.release(key_number)
            else:  # START pressed
                print("START pressed, enter your password")
                macropad.keyboard.release(key_number)
                password_guess = ""
                scramble()
                state = STATE_ENTRY
                continue
        if state == STATE_ENTRY:
            if key_number == 9:  # Start key during entry
                print("Restart whole key entry")
                macropad.keyboard.release_all()
```

password_guess = ""  # Reset password entry
scramble()
continue
#
# From here out is password entry, state is KEY_ENTRY
#
if key_number < 11:  # Ignore encoder and lower right button
    old_color = macropad.pixels[key_number]  # Save color of key pressed
    macropad.pixels[key_number] = 0xFFFFFFFF  # Turn key white while down
    macropad.pixels.show()  # Show key as white
    macropad.play_tone(press_tone, 0.6)  # Play tone when key pressed
# Process input - add the key pressed to the password entry
if key_number == 10:  # The "0" position is shifted over, take one away
    password_guess = password_guess + str(key_values[key_number-1])
else:  # The 1-9 keys (index values 0 to 8)
    password_guess = password_guess + str(key_values[key_number])
print(password_guess)
if len(password_guess) == PASSWORD_LENGTH:  # We've entered all digits
    keys_clear()  # Clear the keypad
    if password_guess == PASSWORD:  # Success
        group[9].text = " "
        group[11].text = "OPEN"
        macropad.display.show(group)
        macropad.display.refresh()
        macropad.pixels.show()
    else:  # fail!
        group[11].text = "FAIL"
        group[9].text = " "
        macropad.display.show(group)
        macropad.display.refresh()
for _ in range(3):  # Flash lower right 3 times red with beeps
    macropad.pixels.show()
    macropad.play_tone(880, 1)
    time.sleep(0.1)
    macropad.pixels.show()
    time.sleep(0.1)
# Reset state after both success and failure
keys_clear()
state = STATE_RESET
else:  # Release any still-pressed keys
    macropad.keyboard.release(key_number)
# Change key color back
if state == STATE_ENTRY:
    if key_number in (0, 1, 2, 3, 4, 5, 6, 7, 8, 10):
        macropad.pixels[key_number] = old_color
        macropad.pixels.show()

Passcode Changes and Security

The passcode is coded into the CircuitPython program, the default is "2468". You can change this by editing the code.py file with Mu or another text file editor. The
passcode must be composed of digits 0 to 9 and may be of any length greater than one.

Note that embedding the passcode in the file is not the most secure. The best method would be to read it from a file, such as secrets.py or even some encrypted methodology. This project does not go to great lengths for this demonstration and the user may augment their own program to provide passcode security, multiple passcodes, and other improvements.

Files

The CIRCUITY drive should have the following files in the root directory and lib folder.

Use

The project works much like a Hirsch Scramblepad but with a couple of simplifications. Here is the operation:

When the MacroPad is powered up via the USB port, the title bar on the display will say Scramblepad. The spaces below correspond to the 12 keys in the keypad. The program will label those appropriately during each mode of operation.
START Mode

The mode where there is only the word START in the lower left corner is the initial starting mode. No keypad input works except the lower left key. That must be pressed to enable the ability to enter a passcode.

I used black windowed lamp keycaps (https://adafruit.it/UC0) for this key and for the lower right key. Lower right only indicates open and closed, it is not used to enter data). Differentiating these keys with darkened, windows keys helps with understanding operation. If you could find a key that actually says START it'd be a huge bonus.
ENTRY Mode

Once the START key has been pressed, the keyboard will flash and random sequences of keys will be shown on the display three times.

Finally, ten numbers will be on the display, corresponding to the lights on the keypad.

The user is expected to enter the password one digit at a time, looking at the display to see where the next number is and pushing the corresponding key on the keypad. Do not assume the keys are numbered like a phone dialing pad, this is scrambled input.

OPEN Mode

When the passcode is entered correctly, the lower right key will turn green and OPEN will be on the display for that key. If you have connected a solenoid per the Lock Circuit page, the lock should click open for 2 seconds.

After 7 seconds the lock will go back to START Mode.

FAIL Mode

If after the number of digits in the passcode the wrong passcode is entered, the lower left key will be marked FAIL and blink red 3 times. Then the keypad will go back to the START Mode.

Pressing START during ENTRY

If you wish to start over entering a passcode, press the lower left START key at any time. The keypad will scramble 3 times and you can try again noting the numbers in their new locations.