Lockbox Algorithm, Design, and Final Code: Complete Project Documentation

Project Overview

This document provides the full details for designing, building, and programming a secure physical lockbox system. The project combines a binary tree button sequence with a 3x3 numeric keypad to create a two-phase authentication system controlled by a Raspberry Pi. The document includes hardware requirements, system functionality, security justification, reset features, and the complete final working code.

System Phases and Process

Phase 1: Binary Tree Button Sequence

• 31 pushbuttons arranged to represent a 5-level binary tree.
• 31 LEDs provide feedback for each correct input.
• User must press the correct sequence of buttons based on valid tree paths.
• Incorrect input resets the sequence and all LEDs.

Phase 2: Numeric Keypad PIN Entry

• After completing the binary tree sequence, user enters a PIN code on a 3x3 numeric keypad.
• Correct PIN disengages the lock using a servo motor.
• Incorrect PIN resets both authentication phases.

Hardware Requirements

• Raspberry Pi with sufficient GPIO pins
• 31 pushbuttons and corresponding 31 LEDs with resistors
• 3x3 numeric keypad (digits 1-9)
• High-torque servo motor for mechanical lock
• Wiring, connectors, breadboard or PCB
• Durable lockbox enclosure with space for components

Combination Possibilities and Security Strength

The system provides 32 unique valid button paths in the binary tree sequence.
The 3x3 keypad allows 9^4 = 6,561 possible PIN code combinations.
Total combination space: 32 × 6,561 = 209,952 possible access sequences.
The system is physically isolated with no wireless interfaces, preventing conventional digital hacking attempts.
Only authorized physical input sequences can trigger unlocking.

System Reset and Password Management

Authorized users can access internal system settings to reset both the binary tree sequence and the keypad PIN. Physical access to the control hardware is required to perform resets, ensuring tamper resistance.

System Logic and Security Justification

The lockbox operates as a finite state machine (FSM), advancing only on correct inputs. No remote connections, external scanning, or wireless methods can compromise the system. Physical tampering is deterred by mechanical reinforcement and secure construction.

Startup Initialization

On startup, the system:
• Initializes all GPIO pins.
• Resets all LEDs.
• Clears input progress counters.
• Waits for valid binary tree inputs to begin authentication.

Final Python Code

```
import RPi.GPIO as GPIO
import time

button_pins = list(range(2, 33))  # 31 pushbuttons
led_pins = list(range(33, 64))    # 31 LEDs
keypad_row_pins = [64, 65, 66]
keypad_col_pins = [67, 68, 69]
servo_pin = 70

correct_path = [0, 1, 3, 7, 15]   # Example correct sequence
```

```
correct_pin = "1234"
current_step = 0
unlocked = False

keypad_keys = [
    ['1', '2', '3'],
    ['4', '5', '6'],
    ['7', '8', '9']
]

def setup():
    GPIO.setmode(GPIO.BCM)
    for pin in button_pins:
        GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    for pin in led_pins:
        GPIO.setup(pin, GPIO.OUT)
    for pin in keypad_row_pins + keypad_col_pins:
        GPIO.setup(pin, GPIO.OUT)
    GPIO.setup(servo_pin, GPIO.OUT)
    reset_leds()

def reset_leds():
    for led in led_pins:
        GPIO.output(led, GPIO.LOW)

def check_tree_buttons():
    global current_step, unlocked
    for idx, pin in enumerate(button_pins):
        if GPIO.input(pin):
            time.sleep(0.1)
            if idx == correct_path[current_step]:
                GPIO.output(led_pins[idx], GPIO.HIGH)
                current_step += 1
                if current_step == len(correct_path):
                    unlocked = True
            else:
                reset_leds()
                current_step = 0

def scan_keypad():
    key_press = None
    for c in range(3):
        GPIO.setup(keypad_col_pins[c], GPIO.OUT)
        GPIO.output(keypad_col_pins[c], GPIO.LOW)

        for r in range(3):
            GPIO.setup(keypad_row_pins[r], GPIO.IN, pull_up_down=GPIO.PUD_UP)
            if GPIO.input(keypad_row_pins[r]) == GPIO.LOW:
                key_press = keypad_keys[r][c]
                time.sleep(0.2)
                break

        GPIO.setup(keypad_col_pins[c], GPIO.IN, pull_up_down=GPIO.PUD_UP)
        if key_press:
            break

    return key_press

def unlock_safe():
    servo = GPIO.PWM(servo_pin, 50)
    servo.start(7.5)
    servo.ChangeDutyCycle(12.5)
    time.sleep(1)
    servo.ChangeDutyCycle(7.5)
    servo.stop()

setup()
try:
    entered_pin = ""
    while True:
        if not unlocked:
            check_tree_buttons()
        else:
            print("Binary tree sequence correct. Enter PIN:")
            while len(entered_pin) < 4:
                key = scan_keypad()
                if key:
                    print(f"Pressed: {key}")
                    entered_pin += key
            if entered_pin == correct_pin:
                print("Correct PIN. Unlocking safe.")
                unlock_safe()
                break
            else:
                print("Incorrect PIN. Resetting system.")
                reset_leds()
                entered_pin = ""
                current_step = 0
                unlocked = False

except KeyboardInterrupt:
    print("System terminated.")
finally:
    GPIO.cleanup()
```