A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A computer screen shot of a black screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**⚠️ Issues / Suggestions**

**🔧 Issues:**

* pass1, pass2, etc. are declared but unused.
* SP\_PIN is defined but unused.
* No timeout or retry limit for failed attempts.
* Servo reads potentiometer instead of just rotating to a fixed unlock angle — this is likely for testing but impractical in real usage.

**💡 Improvements:**

* Store multiple RFID UIDs in an array for multiple users.
* Store/change passcodes using EEPROM.
* Use a fixed angle instead of potentiometer to control servo.
* Add retry limits and lockout timer for brute-force protection.
* Mask passcode on LCD with \* instead of showing digits.

**🧠 Software Challenges**

**1. Incorrect UID Handling**

* Parsing and comparing RFID UIDs improperly can lead to false rejections.
  + 🔧 **Fix**: Use uppercase, consistent formatting (F4:53:9E:22), and avoid hidden characters.

**2. Keypad Debouncing**

* Pressing a key may register multiple times or skip due to bounce.
  + 🔧 **Fix**: Use built-in debounce in the Keypad library and brief delays.

**3. Blocking Delays**

* delay() functions block the whole system, which can affect responsiveness.
  + 🔧 **Fix**: Use millis()-based non-blocking delays for better user experience.

**4. String Comparison Bugs**

* Using == instead of .equals() or strcmp() incorrectly can break authentication.
  + 🔧 **Fix**: Stick to strncmp() or .equalsIgnoreCase() where applicable.

**👤 User Experience Challenges**

**1. Lack of Feedback**

* Users may not know if they’ve entered a wrong key or scanned an invalid card.
  + 🔧 **Fix**: Use the buzzer, LCD, and LEDs for immediate feedback.

**2. Password Entry Mistakes**

* If the user mistypes, there's no way to delete or restart input.
  + 🔧 **Fix**: Add \* to clear the entry and # to submit.

**3. No Timeout or Reset**

* If someone starts to enter a code and stops midway, the system waits forever.
  + 🔧 **Fix**: Add a timeout (e.g., 10 seconds of inactivity) to cancel the process.

**🔐 Security Challenges**

**1. Hardcoded Credentials**

* UID and passcode are stored in plain text in the code.
  + 🛡️ **Fix**: Move credentials to EEPROM and provide a secure way to update them.

**2. Brute-force Vulnerability**

* Anyone can guess passcodes endlessly with no lockout mechanism.
  + 🛡️ **Fix**: Add retry limits and timed lockouts after failed attempts.

**3. RFID Spoofing**

* Some cheap RFID cards can be easily cloned.
  + 🛡️ **Fix**: Use more secure RFID types (e.g., MIFARE DESFire) or multi-factor authentication.

**4. No Logging**

* No record of who accessed the system and when.
  + 🛡️ **Fix**: Use an SD card or send logs via serial/Wi-Fi for auditing.

**🌐 Optional Expansion Risks**

If you decide to add:

* **IoT/Wi-Fi (e.g., ESP8266)** → Connectivity bugs, power draw
* **App integration** → Auth token management, app security
* **Remote unlock** → You need secure encryption, SSL, and validation

**🔒 Secure It By:**

* Using a basic encryption like:
  + XOR-based
  + Vigenère cipher
  + Lightweight AES library (on bigger MCUs)
* Obfuscating file formats

**☁️ 5. Cloud-based Credential Management (IoT)**

**📌 What It Is:**

Use a Wi-Fi-enabled board (ESP32, ESP8266, etc.) and authenticate via a cloud database (Firebase, Google Sheets, custom API).

**✅ Pros:**

* Dynamic, real-time credential updates
* Centralized control (ideal for multiple access points)
* Scalable to many users

**⚠️ Cons:**

* Needs internet access
* Adds latency and more points of failure
* Requires backend security to protect API endpoints

**🔒 Secure It By:**

* HTTPS communication
* API keys or OAuth
* Rate limiting and access logging