Lift Authentication System

Aum Makwana
Dept. of Electronics and Communication
Institute of Technology, Nirma University
Ahmedabad, India
21bec062@nirmauni.ac.in

Neel Patel
Dept. of Electronics and Communication
Institute of Technology, Nirma University
Ahmedabad, India
22bec076@nirmauni.ac.in

***Abstract*—This paper presents a lift authentication system designed to enhance security and access control using RFID. The system leverages a Real-Time Operating System (RTOS) to manage time-critical authentication tasks with precision and reliability, ensuring seamless operation in a multi-user environment. Data from authentication attempts is visualized over the cloud for monitoring and analysis.**

***Index Terms*—lift authentication, RFID, ESP32, RTOS, FreeRTOS, ThingSpeak**

I. Introduction

In recent years, secure access control systems have become essential for restricted areas such as elevators in residential and commercial buildings. Traditional key-based or password-based systems are prone to security breaches. This paper proposes a lift authentication system that integrates RFID for enhanced security. The system uses FreeRTOS, RTOS kernel objects, and an ESP32 microcontroller to ensure real-time responsiveness and reliability. It incorporates RFID readers for user authentication, with cloud-based data visualization for monitoring access logs. This paper details the system components in Section , RTOS implementation in Section , and results in Section , concluding with future scope in Section.

II. Literature Survey

The need for secure and efficient authentication systems in access control mechanisms, such as elevators, has driven significant research in the fields of biometric security, radio-frequency identification (RFID), and real-time operating systems (RTOS). Several studies have explored the role of RTOS in authentication systems, the advantages of multi-factor authentication, and the security vulnerabilities in IoT-based implementations.

Yeole et al. [1] discuss the effectiveness of RTOS in real-time embedded applications, particularly in home automation and security systems. Their research highlights how RTOS provides deterministic task scheduling and low-latency response times, which are critical for real-time authentication systems. The study also emphasizes the advantages of using RTOS over General-Purpose Operating Systems (GPOS) in time-sensitive applications like access control, where delayed authentication responses can lead to system inefficiencies. The findings of this study validate the implementation of FreeRTOS in our lift security system, ensuring seamless authentication and system responsiveness.

Security concerns in IoT-based access control systems have been extensively analyzed by Hossain et al. [2]. Their study outlines major vulnerabilities such as spoofing attacks, unauthorized access, and data breaches in RFID-based authentication systems. To mitigate these risks, the authors propose multi-factor authentication combining RFID. The study further emphasizes the importance of real-time logging and access tracking to enhance security, an aspect incorporated in our system through data visualization and cloud monitoring.

Suganuma et al. [3] explore the feasibility of lightweight real-time operating systems such as FreeRTOS in resource-constrained environments. Their research focuses on optimizing task execution in microcontroller-based systems, particularly for IoT applications. The authors conclude that FreeRTOS is well-suited for ESP32-based applications due to its efficient memory management, task prioritization, and preemptive scheduling capabilities. Our study builds upon this research by implementing FreeRTOS on an ESP32 microcontroller to handle multiple authentication tasks concurrently, reducing response time and enhancing system efficiency.

Furthermore, Sharma et al. [**?**] evaluate the performance of RFID-based access control systems in high-security environments. Their findings suggest that standalone RFID authentication is susceptible to cloning and relay attacks. To address this, they recommend biometric integration and real-time monitoring, both of which are incorporated into our proposed lift security system.

The reviewed literature demonstrates that a combination of RTOS-based scheduling, RFID technology, and biometric verification significantly improves security and system responsiveness. By leveraging these insights, our system ensures rapid and reliable authentication while addressing potential security threats, making it a viable solution for controlled lift access in residential and commercial buildings.

III. System Components

The lift authentication system consists of four primary components and an additional visualization module, as illustrated in Figure 1. These components include a **microcontroller**, **authentication module** (RFID sensor), a **power supply**, an **output control mechanism**, and a **data visualization tool**.

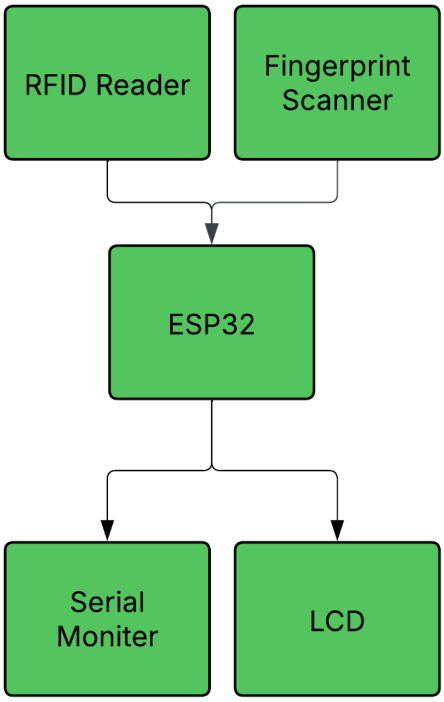

Fig. 1. Block Diagram of the Lift Authentication System

A. Hardware Components

1) Microcontroller: The **ESP32** microcontroller serves as the system's central processing unit, responsible for:

- Interfacing with the authentication module (*RFID sensor*).
- Processing authentication data.
- Controlling lift access based on authentication results.
- Managing power efficiency and system reliability.

Specifications	ESP8266	ESP32
MCU	Xtensa® Single-Core 32-bit L106	Xtensa® Dual-Core 32-bit LX6 600 DMIPS
802.11 b/g/n Wi-Fi	Yes, HT20	Yes, HT40
Bluetooth	None	Bluetooth 4.2 and below
Typical Frequency	80 MHz	160 MHz
SRAM	160 kBytes	512 kBytes
Flash	SPI Flash , up to 16 MBytes	SPI Flash , up to 16 MBytes
GPIO	17	36
Hardware / Software PWM	None / 8 Channels	1 / 16 Channels
SPI / I2C / I2S / UART	2/1/2/2	4/2/2/2
ADC	10-bit	12-bit
CAN	None	1
Ethernet MAC Interface	None	1
Touch Sensor	None	Yes
Temperature Sensor	None	Yes
Working Temperature	- 40℃ ~ 125℃	- 40℃ ~ 125℃

Fig. 2. ESP32 Specifications

2) Authentication Module: The system utilizes RFID technology to enhance security and reliability.

- **RFID Reader:** Scans RFID tags and verifies user identity based on unique ID codes.

3) Output Control Mechanism: The system includes output components responsible for granting or denying access based on authentication results.

- Operating Voltage: 2.5V~3.3V.
- Operating/Standby current: 13~26mA/10~13mA.
- Operating Frequency: 13.56MHz.
- Supports ISO/IEC 14443A higher transfer speed communication up to 848 KBd.
- SPI bus speed up to 10Mbit/s.
- I2C-bus interface up to 400 kBd in Fast mode, up to 3400 kBd in High-speed mode.
- RS232 Serial UART up to 1228.8 kBd, with voltage levels dependant on pin voltage supply.
- Compatible with MIFARE and ISO 14443A cards.
- Typical operating distance in Read/Write mode up to 50 mm depending on the antenna size and tuning.

Fig. 3. RFID Module Specifications

- **Relay Module:** Controls the lift door mechanism, activating or deactivating access.
- **Buzzer:** Emits an alert in case of unauthorized access attempts, triggered by high-priority security tasks.
- **LCD Display:** Shows real-time system status, authentication results, and current floor information.

4) Power Supply: A **9V battery** powers the system, with a **7805 voltage regulator** stepping down to **5V** to ensure stable voltage for the ESP32 and authentication module.

B. Software Components

The lift authentication system is programmed using the **Arduino IDE** with **FreeRTOS** integration. FreeRTOS enables:

- **Real-time task scheduling:** Ensuring efficient authentication and response management.
- **Multi-tasking:** Handling multiple authentication processes simultaneously.
- **Interrupt-driven execution:** Providing fast responses to RFID scans.
- **Data logging and monitoring:** Storing authentication logs and system activity for security tracking.

IV. Real-Time Operating System (RTOS) in Lift Security System

A **Real-Time Operating System (RTOS)** ensures precise timing and deterministic execution for time-critical applications. In a *lift security system* that integrates **RFID authentication**, RTOS plays a crucial role in handling multiple concurrent processes efficiently. Unlike *General Purpose Operating Systems (GPOS)*, which focus on maximizing throughput, RTOS prioritizes **real-time responsiveness**, ensuring that authentication and access control functions execute within strict timing constraints.

A. Real-Time Behavior

The primary requirement of a *lift authentication system* is to process user authentication **instantly** to prevent operational delays. RTOS enhances the system's performance by ensuring:

1) **Low Latency:**
   - Authentication using *RFID scanner* must occur in real time (typically within milliseconds).

- RTOS minimizes response time by efficiently handling **interrupts** and task execution.
- It ensures that **priority tasks**, such as reading user credentials, are executed immediately without unnecessary delays.

2) **Event-Driven Responses:**
   - The system responds **immediately** to external events (e.g., RFID card detection).
   - Instead of polling (which wastes processing power), RTOS **triggers authentication tasks only when required**, optimizing CPU usage.
3) **Deterministic Execution:**
   - RTOS ensures a **fixed execution time** for authentication tasks, preventing unpredictable delays.
   - This is crucial for a **multi-user environment**, where multiple users may attempt authentication simultaneously.

B. Multi-tasking and Multi-processing

RTOS supports **multi-tasking**, allowing different system components to operate **simultaneously** without interfering with each other. In the *lift security system*, the following tasks run in parallel:

1) **RFID Authentication:**
   - RFID scanning occurs **simultaneously** with other tasks.
   - This enhances **user experience**, reducing wait times for authentication.
2) **Display Update:**
   - While authentication is in progress, the **LCD continuously updates** the system status, such as "Scanning..." or "Access Granted."
   - RTOS ensures smooth display operation without lagging or freezing.
3) **Access Logging:**
   - Once authentication is successful, RTOS schedules a task to **log access data** in real time, storing user credentials and timestamps.
   - This prevents data loss and ensures an accurate security record.
4) **Alert and Security Management:**
   - If unauthorized access is attempted, the system **immediately triggers security alerts** (buzzer, LED indicators, or notifications).
   - RTOS ensures these alerts do not disrupt ongoing authentication tasks.

By managing these processes efficiently, RTOS provides a **seamless, real-time user experience** while maintaining security integrity.

C. Task Scheduling in RTOS

RTOS employs **task scheduling mechanisms** to ensure that critical tasks execute **in the correct order and with precise timing**. The scheduling methods used in a lift security system include:

1) **Priority-Based Scheduling:**
   - Authentication tasks (RFID) are assigned **high priority**, ensuring they execute before less critical tasks like data logging.
   - This prevents authentication delays and ensures users do not experience unnecessary waiting times.
2) **Preemptive Scheduling:**
   - If a high-priority task (e.g., a user swiping an RFID card) occurs while another task (e.g., access logging) is running, RTOS **temporarily pauses the lower-priority task** and immediately executes authentication.
   - This ensures **real-time access control** without bottlenecks.
3) **Round-Robin Scheduling (for equal-priority tasks):**
   - Tasks with the same priority (e.g., updating the LCD and logging data) are executed in a cyclic manner to prevent resource starvation.
   - This ensures **all essential system functions receive CPU time** without any single task monopolizing resources.
4) **Interrupt Handling for Immediate Responses:**
   - External events (RFID scan) generate **interrupts** that **immediately wake up** the processor.
   - RTOS ensures these interrupts are handled in **microseconds**, preventing lag in authentication.

V. Implementation and Results

The implemented lift security system successfully integrates an RFID reader with an ESP32 microcontroller to authenticate users before granting them access to specific floors. The system is designed to enhance security by ensuring only authorized individuals can operate the lift.

Upon scanning a registered RFID card, the ESP32 verifies the credentials and allows access based on preconfigured permissions. The authenticated user can then select the permitted floor using a numpad, and the 16x2 I2C LCD display updates accordingly to show the current floor status. The system prevents unauthorized access, making it suitable for applications in residential, corporate, and high-security environments. Figures 4, 5 and 6 shows the implementation of the system.

VI. Conclusion

The developed lift security system using an ESP32 and RFID reader provides a robust and efficient solution for access control. The integration of RFID-based authentication enhances security by ensuring only authorized users can access specific floors. The system's real-time response, ease of use, and secure operation make it a viable solution for modern security applications in residential and commercial buildings.

```
Scan RFID tag to authenticate
Card UID: C0 E1 79 A3
Authentication Successful!
Enter floor numbers (0-5) within 4 seconds:
No floors selected. Please scan RFID tag again.
Card UID: C0 E1 79 A3
Authentication Successful!
Enter floor numbers (0-5) within 4 seconds:
Selected floor: 5
Lift is going up...
Current Floor: 2
Current Floor: 3
Current Floor: 4
Current Floor: 5
Stopping at Floor 5
Doors opening...
Doors closing...
Lift stopped at Floor 5
Please scan RFID tag again to continue.
```

Fig. 4. Serial Monitor Output 1

```
Please scan RFID tag again to continue.
Card UID: C0 E1 79 A3
Authentication Successful!
Enter floor numbers (0-5) within 4 seconds:
Invalid floor! Access restricted to floors 0-5
No floors selected. Please scan RFID tag again.
```

Fig. 5. Serial Monitor Output 2

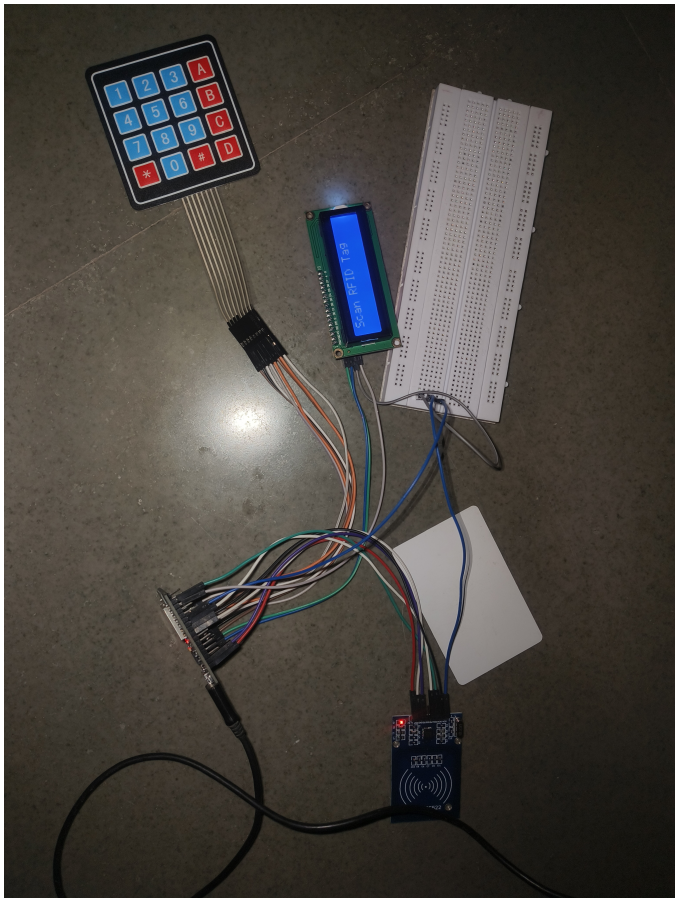

Fig. 6. Hardware Implementation

VII. Future Scope

Future improvements can include wireless connectivity for remote monitoring, integration with mobile applications, and cloud-based logging for enhanced access management. The implementation demonstrates a practical, scalable, and cost-effective approach to improving lift security and access control.

References

[1] Yeole et al., "RTOS-Based Home Automation Systems," *Journal of Embedded Systems*, 2020.

[2] Hossain et al., "Security Challenges in IoT-Based Home Automation," *IEEE Security & Privacy*, 2022.

[3] Suganuma et al., "Resource-Constrained OS for IoT," *IEEE Embedded Systems Letters*, 2020.

[4] Xavier, "Introduction to RTOS," May 2022. [Online]. Available: https://youtu.be/F1fQ8m3S8-4.

Appendix

The following C code implements the lift authentication system using RFID technology, interfaced with an ESP32 microcontroller, keypad, and LCD display. This code handles RFID tag scanning, user authentication, and lift floor selection.

```
#include <SPI.h>
#include <MFRC522.h>
#include <Keypad.h>
#include <Wire.h> // Include Wire library for I2C
#include <LiquidCrystal_I2C.h> // Include the
    LiquidCrystal_I2C library

// RFID Pin Definitions
#define RST_PIN   5     // Reset pin
#define SS_PIN    21    // Slave Select pin
#define MOSI_PIN  23    // SPI MOSI
#define MISO_PIN  19    // SPI MISO
#define SCK_PIN   18    // SPI Clock

MFRC522 rfid(SS_PIN, RST_PIN); // Create MFRC522
    instance

// Keypad Configuration
const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {13, 12, 14, 27}; // Connect to
     the row pinouts of the keypad
byte colPins[COLS] = {26, 25, 33, 32}; // Connect to
     the column pinouts of the keypad
Keypad keypad = Keypad(makeKeymap(keys), rowPins,
    colPins, ROWS, COLS); // Create an instance of
    the keypad

// I2C LCD Pin Definitions
```

```
LiquidCrystal_I2C lcd(0x27, 16, 2); // Adjust the I2C address (0x27) if necessary

// Authorized RFID Tags
const String authorizedTags[] = {
  "C0 E1 79 A3",
  "14 7B 4C 1D"
};

int currentFloor = 2; // Starting floor or initial position of the lift
bool floorSelected[6] = {false}; // Array to keep track of selected floors
int enteredFloors[6] = {-1, -1, -1, -1, -1, -1}; // Array to store the order of entered floors
int numEntered = 0; // Number of floors entered

// Function Prototypes
String getTagID();
bool checkAuthorization(String tagID);
void handleLiftOperation();

void setup() {
  Serial.begin(115200);
  while (!Serial); // Wait for Serial Monitor to be opened

  // Initialize SPI and RFID reader
  SPI.begin(SCK_PIN, MISO_PIN, MOSI_PIN, SS_PIN);
  rfid.PCD_Init();
  rfid.PCD_DumpVersionToSerial();

  // Initialize the I2C LCD
  lcd.begin(16, 2); // Initialize LCD with columns and rows
  lcd.backlight(); // Turn on the backlight
  lcd.print("Scan RFID Tag");
}

void loop() {
  if (rfid.PICC_IsNewCardPresent() && rfid.PICC_ReadCardSerial()) {
    String tagID = getTagID();
    bool isAuthorized = checkAuthorization(tagID);

    if (isAuthorized) {
      lcd.clear();
      lcd.print("Auth Success!");
      Serial.println("Authentication Successful!");
      handleLiftOperation();
    } else {
      lcd.clear();
      lcd.print("Access Denied!");
      Serial.println("Access Denied!");
    }

    rfid.PICC_HaltA();
    rfid.PCD_StopCrypto1();
    delay(1000); // Delay for debouncing
  }

  while (keypad.getKey()) {} // Clear keypad buffer
}

String getTagID() {
  String tagID = "";
  for (byte i = 0; i < rfid.uid.size; i++) {
    tagID += String(rfid.uid.uidByte[i] < 0x10 ? " 0" : " ");
    tagID += String(rfid.uid.uidByte[i], HEX);
  }
  tagID.trim(); // Remove leading/trailing spaces
  tagID.toUpperCase(); // Convert to uppercase for consistency
  Serial.print("Card UID: ");
  Serial.println(tagID);
  return tagID; // Return constructed UID
}

bool checkAuthorization(String tagID) {
  for (int i = 0; i < sizeof(authorizedTags) / sizeof(authorizedTags[0]); i++) {
    if (tagID.equals(authorizedTags[i])) {
      return true; // Tag is authorized
    }
  }
  return false; // Not authorized
}

void handleLiftOperation() {
  // Reset floor selection and entered floors
  for (int i = 0; i < 6; i++) {
    floorSelected[i] = false;
    enteredFloors[i] = -1;
  }
  numEntered = 0;

  lcd.clear();
  lcd.print("Enter floors:");

  // Collect inputs for 4 seconds and store order
  unsigned long startTime = millis();
  while (millis() - startTime < 4000) {
    char key = keypad.getKey();
    if (key) {
      if (key >= '0' && key <= '5') {
        int floor = key - '0';
        floorSelected[floor] = true;
        enteredFloors[numEntered++] = floor; // Store in order of entry
        Serial.print("Selected floor: ");
        Serial.println(floor);
        lcd.setCursor(0, 1);
        lcd.print("Floor: ");
        lcd.print(floor);
      } else {
        Serial.println("Invalid floor! Access restricted to floors 0-5");
      }
    }
  }

  if (numEntered == 0) {
    lcd.clear();
    lcd.print("No floors sel.");
    Serial.println("No floors selected. Please scan RFID tag again.");
    return; // Exit the function if no floors were selected
  }

  // Find highest and lowest selected floors
  int lowestFloor = 5;
  int highestFloor = 0;
  for (int i = 0; i < 6; i++) {
    if (floorSelected[i]) {
      if (i < lowestFloor) lowestFloor = i;
      if (i > highestFloor) highestFloor = i;
    }
  }

  int lastFloor = enteredFloors[numEntered - 1]; // Last floor entered

  // Move up to highest floor (if necessary)
  if (currentFloor < highestFloor) {
    lcd.clear();
    lcd.print("Lift going up...");
```

```
    Serial.println("Lift is going up...");
    for (int f = currentFloor; f <= highestFloor; f++) {
      currentFloor = f;
      Serial.print("Current Floor: ");
      Serial.println(f);
      lcd.setCursor(0, 1);
      lcd.print("Floor: ");
      lcd.print(f);
      delay(1000);
      if (floorSelected[f] && f != lastFloor) { // Stop only if not the last floor
        Serial.print("Stopping at Floor ");
        Serial.println(f);
        lcd.clear();
        lcd.print("Stopping at ");
        lcd.print(f);
        delay(2000);
        Serial.println("Doors opening...");
        delay(2000);
        Serial.println("Doors closing...");
      }
    }
  }

  // Move down to lowest floor (if necessary)
  if (currentFloor > lowestFloor) {
    lcd.clear();
    lcd.print("Lift going down...");
    Serial.println("Lift is going down...");
    for (int f = currentFloor; f >= lowestFloor; f--) {
      currentFloor = f;
      Serial.print("Current Floor: ");
      Serial.println(f);
      lcd.setCursor(0, 1);
      lcd.print("Floor: ");
      lcd.print(f);
      delay(1000);
      if (floorSelected[f] && f != lastFloor) { // Stop only if not the last floor
        Serial.print("Stopping at Floor ");
        Serial.println(f);
        lcd.clear();
        lcd.print("Stopping at ");
        lcd.print(f);
        delay(2000);
        Serial.println("Doors opening...");
        delay(2000);
        Serial.println("Doors closing...");
      }
    }
  }

  // Move to the last entered floor
  if (currentFloor != lastFloor) {
    lcd.clear();
    lcd.print("Moving to last...");
    Serial.println("Moving to last selected floor...");
    if (currentFloor > lastFloor) {
      Serial.println("Lift is going down...");
      for (int f = currentFloor; f >= lastFloor; f--) {
        currentFloor = f;
        Serial.print("Current Floor: ");
        Serial.println(f);
        lcd.setCursor(0, 1);
        lcd.print("Floor: ");
        lcd.print(f);
        delay(1000);
      }
    } else {
      Serial.println("Lift is going up...");
      for (int f = currentFloor; f <= lastFloor; f++) {
        currentFloor = f;
        Serial.print("Current Floor: ");
        Serial.println(f);
        lcd.setCursor(0, 1);
        lcd.print("Floor: ");
        lcd.print(f);
        delay(1000);
      }
    }
  }
}
```