**FORMAN CHRISTIAN COLLEGE (A CHARTERED UNIVERSITY)**

**A blue and white logo

Description automatically generated**

**CSCS306 - A**

**FA24**

**Lab 4**

**Hafsah Shahbaz – 251684784**

**Daim Bin Khalid** **– 251686775**

**Syeda Manal Ammad – 251606966**

## Introduction

In this lab, we explored key cryptographic techniques by implementing hashing and encryption algorithms using Arduino. The lab involved two tasks: the first task focused on MD5 hashing, where we securely stored user credentials and compared them with user input to verify access. The second task involved XOR-based encryption, converting plain text into encrypted data and then decrypting it to its original form. These tasks provided practical insight into security mechanisms used in embedded systems, enhancing our understanding of data integrity and confidentiality in resource-constrained environments.

## Functions and Libraries

In this lab, we utilized several built-in libraries and custom functions to achieve the objectives of hashing and encryption. Below is a breakdown of the functions and libraries used in each task.

### Task 1 (MD5 Hashing)

* **Libraries Used:**
  + *MD5 Library*: This library was used to hash passwords with the MD5 algorithm. It ensures that sensitive information such as passwords is not stored in plain text.
  + *Arduino Core Functions*: Serial communication functions such as Serial.begin(), Serial.print(), and Serial.read() were used to interact with the user and the Arduino Serial Monitor.
* **Functions:**
  + setup(): Initializes the serial communication and sets up the environment for the program.
  + getUsername(): Retrieves the user’s name input from the Serial Monitor.
  + getPassword(): Collects the user’s password in text form from the Serial Monitor.
  + hashPassword(): This function applies the MD5 hashing algorithm to the password provided by the user.
  + compareCredentials(): Compares the stored username and hashed password with the entered credentials. If the username and hashed password match, it greets the user; otherwise, it terminates the program with an appropriate error message.

### Task 2 (XOR-Based Encryption)

* **Libraries Used:**
  + *XOR Encryption Library*: This library enables simple encryption and decryption using XOR operations, which are particularly suited for small embedded systems like Arduino.
  + *Arduino Core Functions*: Functions like Serial.print(), Serial.read(), and Serial.begin() were also employed here for user interaction.
* **Functions:**
  + setup(): Initializes the serial communication and sets up the environment for the encryption task.
  + getUserString(): Retrieves a string from the user via the Serial Monitor.
  + encryptString(): Encrypts the input string by applying the XOR operation to each character in the string, resulting in an encrypted output.
  + decryptString(): Decrypts the previously encrypted string using the same XOR operation to retrieve the original message.

# Algorithm and Logic

**Task 1: MD5 Hashing**

**Algorithm:**

1. **Setup Phase:**
   * Begin serial communication with the user.
   * Prompt the user to input their username.
   * Prompt the user to input their password.
2. **Hashing Phase:**
   * Use the MD5 algorithm to hash the entered password for secure comparison.
3. **Comparison Phase:**
   * Compare the entered username with the stored username.
   * Compare the hashed password with the stored hashed password.
4. **Output Phase:**
   * If both the username and hashed password match the stored values:
     + Display a "Welcome!" message.
   * If either the username or password is incorrect:
     + Display an "Access Denied!" message.

**Logic:**

* The core logic revolves around hashing the password with MD5 and then comparing it with a pre-stored hashed password. This prevents storing or comparing passwords in plain text, ensuring a higher level of security.

**Task 2: XOR-Based Encryption**

**Algorithm:**

1. **Setup Phase:**
   * Begin serial communication with the user.
   * Prompt the user to input a string to be encrypted.
2. **Encryption Phase:**
   * Encrypt the entered string using the XOR operation with a predefined key.
   * Each character of the input string is XORed with the same key to generate the encrypted string.
3. **Decryption Phase:**
   * Decrypt the encrypted string using the XOR operation with the same key used for encryption.
   * XOR is a reversible operation, so applying XOR again with the same key will retrieve the original string.
4. **Output Phase:**
   * Display both the encrypted and decrypted strings to the user.

**Logic:**

* The XOR encryption method is used to obfuscate data with minimal computational effort. Since XOR is a symmetric operation (i.e., applying it twice with the same key reverses the effect), it is both simple and effective for encryption and decryption in this context.

### Code Breakdown

### Task 1: MD5 Hashing

**Objective:** Implement a system where a user inputs their credentials, and the system checks them against stored hashed credentials using the MD5 hashing algorithm.

1. **setup()**:
   * Initializes the serial communication using Serial.begin(9600) to establish communication between the Arduino and the Serial Monitor.
   * Prompts the user to enter their username using Serial.print() and calls getUsername() to capture it.
   * Prompts the user to enter their password using Serial.print() and calls getPassword() to capture it.
   * Calls hashPassword() to hash the input password.
   * Compares the hashed password with the stored hash by calling compareCredentials().
   * If the credentials match, the program prints a success message; otherwise, it prints an error message and exits.

void setup() {  
 Serial.begin(9600);  
 Serial.println("Enter your username:");  
 String username = getUsername();  
 Serial.println("Enter your password:");  
 String password = getPassword();  
 String hashedPassword = hashPassword(password);  
 compareCredentials(username, hashedPassword);  
}

1. **getUsername()**:

* Waits for the user to input their username on the Serial Monitor.
* Reads the string entered by the user via Serial.readStringUntil('\n') and returns it for further comparison.

String getUsername() {

while (Serial.available() == 0) {} // Wait for input

return Serial.readStringUntil('\n'); // Read and return username

}

1. **getPassword()**:
   * Similar to getUsername(), this function reads the user-entered password from the Serial Monitor and returns it as a string.

String getPassword() {

while (Serial.available() == 0) {} // Wait for input

return Serial.readStringUntil('\n'); // Read and return password

}

1. **hashPassword()**:

* Uses the MD5 library to hash the password string input by the user.
* Converts the password into its MD5 hash equivalent for secure comparison.

String hashPassword(String password) {

return MD5(password); // MD5 hashing function

}

1. **compareCredentials()**:

* This function compares the entered username and hashed password with stored values.
* If both match, it prints a welcome message; otherwise, it prints an error message and denies access.

void compareCredentials(String username, String hashedPassword) {

String storedUsername = "admin";

String storedHashedPassword = "5f4dcc3b5aa765d61d8327deb882cf99"; // MD5 of "password"

if (username == storedUsername && hashedPassword == storedHashedPassword) {

Serial.println("Welcome!");

} else {

Serial.println("Access Denied!");

}

}

### Task 2: XOR-Based Encryption

.

**Objective:** Implement a system that takes a user input string, encrypts it using XOR encryption, and then decrypts it to its original form.

1. **setup()**:
   * Initializes the serial communication and prompts the user to enter a string to be encrypted and decrypted.

void setup() {

Serial.begin(9600);

Serial.println("Enter a string to encrypt:");

String userInput = getUserString()

String encryptedString = encryptString(userInput);

Serial.println("Encrypted String: " + encryptedString);

String decryptedString = decryptString(encryptedString);

Serial.println("Decrypted String: " + decryptedString);

}

1. **getUserString()**:

* Waits for the user to input a string and reads it from the Serial Monitor using Serial.readStringUntil('\n').

String getUserString() {

while (Serial.available() == 0) {} // Wait for input

return Serial.readStringUntil('\n'); // Read and return user input

}

1. **encryptString()**:
   * Encrypts the input string using the XOR operation.
   * Iterates over each character of the string, applies the XOR operation with a predefined key, and returns the encrypted string.

String encryptString(String str) {

char key = 'K'; // XOR key for encryption

String encrypted = "";

for (int i = 0; i < str.length(); i++) {

encrypted += char(str[i] ^ key); // XOR each character

}

return encrypted;

}

1. **decryptString()**:

* Decrypts the previously encrypted string by applying the XOR operation again with the same key.
* Since XOR is reversible, applying XOR with the same key results in the original string being restored.

String decryptString(String str) {

char key = 'K'; // Same XOR key used for decryption

String decrypted = "";

for (int i = 0; i < str.length(); i++) {

decrypted += char(str[i] ^ key); // XOR again to retrieve original message

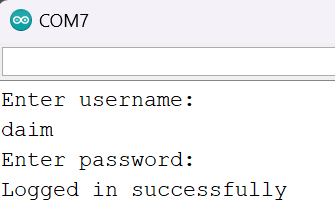
}

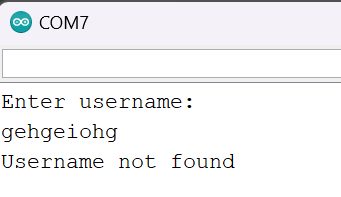
return decrypted;

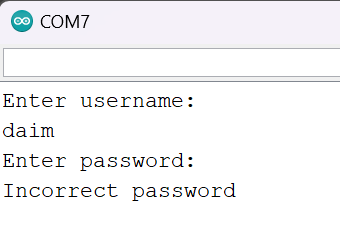
}

### Task 1 Output

The screenshots below show the outcomes for valid and invalid login attempts:







### Task 2 Output

Below is the screenshot showing the encryption and decryption process in action:

A screenshot of a computer

Description automatically generated