Enhancing Trust and Security in IoT Architecture for Low-Cost Microcontroller Devices using Elliptic Curve Cryptography

Kaustubh, Dr. Murthy D H R
Computer Science and Engineering - Cyber Security
Presidency University, Bangalore, India - 560064
E-mail: kaustubh.20201ccs0084@presidencyuniversity.in

Abstract—**In the rapidly expanding universe of the Internet of Things (IoT), low-cost microcontroller devices, such as the NodeMCU, have become ubiquitous. While these devices have transformed how we interact with the physical world, they also introduce significant security vulnerabilities. Chief among these is susceptibility to man-in-the-middle (MITM) attacks, which pose a serious risk to the confidentiality and integrity of data transmitted across these networks. This study addresses this critical issue by proposing the adoption of Elliptic Curve Cryptography (ECC), a cryptographic method known for its efficiency and strong security, despite the computational limitations of low-cost IoT devices. We conducted an experiment demonstrating the ease with which an MITM attack can intercept data from a NodeMCU device. Following this, we implemented ECC to secure data transmission, showcasing its viability as a lightweight yet robust security solution. Our research not only highlights the pressing need for enhanced security measures in the IoT ecosystem but also provides a practical framework for securing low-cost microcontroller devices against sophisticated cyber threats. Through our findings, we contribute to the development of more secure, trustworthy IoT architectures, ensuring that these devices can continue to safely serve as integral components of our digital lives.**

Keywords—NodeMCU, ESP8266, Data theft, Elliptic Curve Cryptography, Man-in-the-middle attack

I. INTRODUCTION

The advent of the Internet of Things (IoT) has ushered in a new era of technology, fundamentally transforming our interaction with the physical world around us. This paradigm shift has been largely facilitated by the proliferation of low-cost microcontroller devices, such as the NodeMCU, which have made IoT technology accessible to a wider range of applications, from home automation to industrial monitoring systems. Despite their widespread adoption, these devices introduce significant security vulnerabilities, with the potential to undermine the integrity, confidentiality, and availability of data.

One of the most pressing concerns in this domain is the susceptibility of IoT devices to Man-in-the-Middle (MITM) attacks. In such attacks, an adversary intercepts communication between devices to eavesdrop or manipulate the data being transmitted. This vulnerability is particularly acute in the context of low-cost microcontroller devices like the NodeMCU, which, while versatile and cost-effective, often lack robust built-in security measures. The ramifications of these security lapses are profound, ranging from the breach of personal privacy to the compromise of critical infrastructure.

Addressing these security challenges is not merely a matter of implementing traditional cryptographic solutions, such as RSA, which, although secure, are computationally intensive and ill-suited to the limited processing power and memory of low-cost IoT devices. This discrepancy between the need for security and the constraints of the devices necessitates the exploration of alternative cryptographic techniques that balance security with computational efficiency.

In this context, Elliptic Curve Cryptography (ECC) emerges as a particularly promising solution. ECC is renowned for its ability to offer strong encryption with smaller key sizes, reducing the computational overhead and making it more applicable for resource-constrained environments like those of the NodeMCU. By leveraging the mathematical properties of elliptic curves, ECC provides a robust security framework, enabling secure communication without imposing undue strain on the device's limited resources.

This research paper aims to address the critical need for enhanced security in low-cost IoT devices by proposing the integration of ECC into the IoT architecture. Through a detailed exploration of the vulnerabilities inherent in these devices, particularly focusing on the NodeMCU's susceptibility to MITM attacks, we underscore the urgency of adopting advanced encryption methods. We present a comprehensive methodology for implementing ECC on the NodeMCU, demonstrating its feasibility and effectiveness in securing data transmission against potential eavesdropping and tampering.

By bridging the gap between the need for security and the operational constraints of low-cost IoT devices, this study contributes significantly to the field of IoT security. It not only highlights the vulnerabilities that threaten the integrity of IoT ecosystems but also introduces a practical, efficient solution to mitigate these risks. Our findings advocate for a reevaluation of security strategies in IoT deployments, emphasizing the importance of adopting encryption techniques like ECC that are both secure and suited to the constraints of low-cost devices. In doing so, this research lays the groundwork for a more secure, trustworthy IoT environment, enabling these technologies to achieve their full potential in enhancing our lives without compromising our safety or privacy.

II. EXPLORING VULNERABILITIES IN LOW-COST MICROCONTROLLER DEVICES

Vulnerability Assessment in Low-Cost IoT Devices

The advent of low-cost microcontroller devices has democratized access to the Internet of Things (IoT), enabling a surge in the development and deployment of IoT applications across various sectors. Devices such as the NodeMCU, built on the ESP8266 Wi-Fi SoC, have become instrumental in this expansion due to their affordability and ease of use. However, the very attributes that make these devices attractive also contribute to their vulnerability to cyber-attacks. This section delves into the inherent vulnerabilities associated with low-cost microcontroller devices, with a focus on the NodeMCU platform.

Data Transmission Vulnerabilities

One of the primary security concerns with low-cost IoT devices is the transmission of sensitive data without adequate encryption. In the case of the NodeMCU device integrated with a BMP280 sensor, data is frequently transmitted to cloud-based platforms such as ThingSpeak for monitoring and analysis. This data, which can include temperature readings, humidity levels, and altitude information, is often sent over the network in plaintext. The lack of encryption not only exposes this information to potential interception but also makes it susceptible to modification and misuse. A study by [1] highlights the prevalence of unencrypted data transmission in IoT devices, underscoring the critical need for secure communication protocols.

The Threat of Man-in-the-Middle (MITM) Attacks

The vulnerability of low-cost IoT devices to Man-in-the-Middle (MITM) attacks is a significant concern. In such attacks, an unauthorized actor intercepts the communication between the device and the server, gaining the ability to read, insert, and modify the messages between the two parties without detection. This type of attack is particularly effective against devices that do not employ secure communication protocols, as demonstrated by [2], where researchers were able to intercept and manipulate data from IoT devices in real-time. The implications of successful MITM attacks range from privacy breaches to the manipulation of device functionality, posing a serious risk to both users and infrastructure.

Inadequate Device Authentication and Authorization

Another critical vulnerability of low-cost IoT devices lies in their often inadequate mechanisms for device authentication and authorization. Without robust authentication protocols, these devices can be easily impersonated, allowing malicious entities to gain unauthorized access to IoT networks. Similarly, insufficient authorization checks can enable attackers to perform actions beyond their permitted scope, further compromising the security of the system. The work by [3] elucidates the challenges in implementing effective authentication and authorization mechanisms in resource-constrained IoT devices, highlighting the trade-off between security and performance.

Limited Computational Resources for Encryption

The constrained computational resources of low-cost microcontroller devices present a significant challenge in implementing traditional encryption algorithms. Cryptographic algorithms like RSA, while secure, require significant computational power and memory, which are scarce resources in devices such as the NodeMCU. This limitation not only impedes the adoption of strong encryption standards but also restricts the device's ability to engage in secure communication, as noted by [4]. The need for lightweight cryptographic solutions that can operate within the constraints of these devices is evident, underscoring the importance of adopting efficient encryption techniques such as Elliptic Curve Cryptography (ECC).

III. PHASE 1: VULNERABILITY ASSESSMENT AND DATA COLLECTION

To investigate the vulnerability of low-cost microcontroller devices and demonstrate the need for enhanced security measures, we conducted a comprehensive vulnerability assessment on a NodeMCU device equipped with a BMP280 sensor for temperature, humidity, and altitude sensing. The objective of this phase was to collect data from the sensor and transmit it to a cloud-based IoT platform, ThingSpeak, without any encryption, thereby exposing the data to potential interception and unauthorized access.

A. *Experimental setup:*

We assembled a NodeMCU ESP8266 development board and connected it to a BMP280 sensor. The sensor was configured to collect temperature, humidity, and altitude data. The NodeMCU board was connected to the internet via its built-in Wi-Fi module, allowing us to transmit the collected data to ThingSpeak.

The BMP280 has SDA and SCL pins that should be connected to D2 and D1 pins on NodeMCU, respectively. The VCC and GND pins should be connected to the 3V3 and G pins on NodeMCU, respectively.

B. *Programming the NodeMCU:*

- Using the Arduino IDE, we developed a code snippet to read data from the BMP280 sensor and establish a connection to the ThingSpeak platform through the Wi-Fi module. The code included necessary libraries, such as Adafruit_BMP280 and ESP8266WiFi, to interface with the sensor and enable Wi-Fi connectivity.

1. *Libraries:*
   - *The code includes the necessary libraries for ESP8266 Wi-Fi functionality (ESP8266WiFi.h) and communication with ThingSpeak (ThingSpeak.h).*
   - *Additionally, it includes libraries for communication with the BMP280 sensor (Wire.h, SPI.h, Adafruit_BMP280.h).*

```
#include <ESP8266WiFi.h>
#include <ThingSpeak.h>

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_BMP280.h>
```

2. *Pin Definitions:*
   - *The code defines the pin connections for the BMP280 sensor (BMP_SCK, BMP_MISO, BMP_MOSI, BMP_CS).*

```
#define BMP_SCK  (13)
#define BMP_MISO (12)
#define BMP_MOSI (11)
#define BMP_CS   (10)
```

3. *Global Variables:*
   - *The code defines the Wi-Fi credentials (ssid and pass).*
   - *It also sets the ThingSpeak channel ID (channelID) and the write API key (writeAPIKey).*

```
Adafruit_BMP280 bmp;

const char *ssid =  "<wifi_ssid>";
const char *pass =  "<wifi_password>";

long channelID = 1234567;
const char* writeAPIKey = "<api_key>";

WiFiClient client;
```

4. *Setup Function:*
   - *The setup() function initializes the serial communication for debugging purposes (Serial.begin()) and checks the BMP280 sensor's presence (bmp.begin()).*
   - *It sets the sampling settings for the sensor using the setSampling() function.*
   - *The ThingSpeak.begin() function initializes the ThingSpeak client with the specified client object.*

```
void setup()
{
  Serial.begin(9600);

  if (!bmp.begin(0x76)) {
    Serial.println(F("Could not find a
valid BMP280 sensor, check wiring!"));
    while (1);
  }

  /* Default settings from datasheet. */

bmp.setSampling(Adafruit_BMP280::MODE_NORM
AL,     /* Operating Mode. */

Adafruit_BMP280::SAMPLING_X2,     /* Temp.
oversampling */

Adafruit_BMP280::SAMPLING_X16,    /*
Pressure oversampling */

Adafruit_BMP280::FILTER_X16,      /*
Filtering. */

Adafruit_BMP280::STANDBY_MS_500); /*
Standby time. */

  ThingSpeak.begin(client);
}
```

5. *Loop Function:*
   - *The loop() function is the main execution loop of the program.*
   - *It checks the Wi-Fi connection status and attempts to connect to the specified SSID and password if not connected.*
   - *Once connected, it reads the temperature, pressure, and altitude from the BMP280 sensor using the corresponding bmp.read...() functions.*
   - *The collected data is then printed to the serial monitor for debugging purposes.*
   - *Next, the data is assigned to specific fields in the ThingSpeak channel using the ThingSpeak.setField() function.*
   - *The data is sent to the ThingSpeak channel using the ThingSpeak.writeFields() function, and the HTTP response code is checked.*
   - *Depending on the write status, a corresponding message is printed to the serial monitor.*
   - *Finally, there is a delay of 15 seconds before the loop restarts.*

```
void loop()
{
  if (WiFi.status() != WL_CONNECTED) {
    Serial.print("Attempting to connect to
SSID: ");
    Serial.println(ssid);
    WiFi.begin(ssid, pass);
    while (WiFi.status() != WL_CONNECTED)
{
      Serial.print(".");
      delay(5000);
    }
    Serial.println("\nConnected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
  }
  float t = bmp.readTemperature();
  float p = bmp.readPressure();
  float a = bmp.readAltitude(1013.25);

  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.println(" deg. C");

  Serial.print("Pressure: ");
  Serial.print(p);
  Serial.println(" Pa");

  Serial.print("Altitude: ");
  Serial.print(a);
  Serial.println(" m");

  Serial.println("Writing data to
ThingSpeak...");

  ThingSpeak.setField(1, t);
  ThingSpeak.setField(2, p);
  ThingSpeak.setField(3, a);

  int writeStatus =
ThingSpeak.writeFields(channelID,
writeAPIKey);

  if (writeStatus == 200)
    Serial.println("Channel updated
successfully. Writing again in 15
seconds...");
  else
    Serial.println("Problem updating
channel. HTTP error code = " +
String(writeStatus) + ". Trying again in
15 seconds...");

  delay(15000);
}
```

C. *Data Transmission to ThingSpeak:*

Once the code was uploaded to the NodeMCU device, it initiated the data collection process from the BMP280 sensor. The collected data, including temperature, humidity, and altitude, was packaged into a message and sent to ThingSpeak using the established Wi-Fi connection. The transmitted data was in plaintext, without any encryption or security mechanisms.

D. *Data Reception and Analysis:*

On the ThingSpeak platform, we created a channel to receive and visualize the data transmitted by the NodeMCU device. We monitored the data reception process to verify the successful transmission of the sensor readings.

E. *Vulnerability Demonstration:*

With the data transmission in progress, we set up a separate machine on the same Wi-Fi network to act as an attacker system. Using popular network analysis tools like Wireshark, we intercepted the network traffic between the NodeMCU device and the ThingSpeak server. Through ARP spoofing techniques, we performed a man-in-the-middle (MITM) attack to capture the unencrypted data packets transmitted by the NodeMCU device.

F. *Data Interception and Analysis:*

By capturing the network traffic, we were able to intercept the unencrypted sensor readings in real-time. We analyzed the intercepted data to demonstrate the severity of the vulnerability. This step emphasized the ease with which an attacker can obtain sensitive information, such as temperature, humidity, and altitude data, from low-cost IoT devices like NodeMCU.

The vulnerability assessment performed in this phase successfully demonstrated the lack of security in the data transmission process of low-cost microcontroller devices. The interception and analysis of unencrypted data highlighted the potential risks and the urgent need for implementing effective security measures. In the subsequent phases of our research, we will propose and implement Elliptic Curve Cryptography (ECC) as a solution to mitigate these vulnerabilities and ensure secure communication between low-cost microcontroller devices and IoT platforms.

IV. PHASE 2: PERFORMING MAN-IN-THE-MIDDLE (MITM) ATTACK ON NODEMCU

In Phase 2 of the research, we will focus on conducting a Man-in-the-Middle (MITM) attack on the NodeMCU device to intercept the traffic between NodeMCU and the server. This experiment aims to highlight the vulnerabilities in the communication channel and emphasize the importance of implementing stronger security measures. The following steps outline the process of performing the MITM attack using mitmproxy CLI tool:

A. *Install Wireshark:*

Install Wireshark on the MacBook or any other computer that will be used as the attacker machine. Wireshark will be used to capture and analyze network traffic.

B. *Connect the Devices:*

Ensure that both the MacBook and NodeMCU are connected to the same Wi-Fi network. This is necessary for successful interception of the traffic.

C. *Download and Install mitmproxy:*

a) Download and install mitmproxy, an open-source tool used for intercepting, modifying, and replaying HTTP/HTTPS traffic.

b) On macOS, mitmproxy can be installed using the Homebrew package manager. Open the terminal and execute the command: brew install mitmproxy.

D. List Devices on the Network:

In the terminal, execute the command arp -a to list all the devices connected to the same network. This will help identify the IP addresses of the devices involved.

E. Enable IP Forwarding:

a) To enable IP forwarding, execute the following command in the terminal:

```
sudo sysctl -w net.inet.ip.forwarding=1
```

b) This step is necessary to allow the MacBook to act as a router for the intercepted traffic.

F. Perform ARP Spoofing:

a) Execute the following command in the terminal to perform ARP spoofing:

```
arpspoof -i en0 -r -t <default gateway>
<victim device IP address>
```

b) Replace <default gateway> with the IP address of the router and <victim device IP address> with the IP address of the NodeMCU device.
c) This step redirects the network traffic from the NodeMCU device to the attacker machine.

G. Enable Port Forwarding:

a) To capture the intercepted traffic, port forwarding needs to be enabled.
b) Execute the following command in the terminal:

```
echo "rdr pass inet proto tcp from any to
any port <port on which data is being
sent> -> 127.0.0.1 port <port on which
mitmproxy is running>" | sudo pfctl -ef -
```

c) Replace <port on which data is being sent> with the specific port number on which the data is being sent from NodeMCU to the server.
d) Replace <port on which mitmproxy is running> with the port number on which mitmproxy is running (typically 8080).

H. Run mitmproxy:

a) Start mitmproxy by executing the following command in the terminal:

```
mitmproxy --mode transparent --showhost -p
<port on which mitmproxy is running> -k
```

b) The --mode transparent flag specifies the attack mode as transparent, meaning the victim (NodeMCU) won't be aware of the attack.
c) The --showhost flag displays the host name of the server.
d) The -p flag specifies the port number on which mitmproxy is running (in this case, 8080).
e) The -k flag disables TLS verification as we are intercepting HTTPS traffic.

I. Intercept Data:

a) At this point, mitmproxy is actively intercepting the traffic between NodeMCU and the server.
b) Monitor the terminal running mitmproxy to observe the data being sent from NodeMCU to the server.
c) Note that since the data is not encrypted, sensitive information can be easily captured during this interception.

J. Revert Port Forwarding Rule (optional):

a) Once the interception is complete, it is important to revert the port forwarding rule to restore normal network traffic flow.
b) Execute the following command in the terminal to revert the port forwarding rule:

```
sudo pfctl -f /etc/pf.conf
```

c) This step is only applicable to macOS. For Linux, use the appropriate iptables command to revert the port forwarding rule.

The successful execution of the MITM attack in Phase 2 sheds light on the critical vulnerabilities that exist in the communication between NodeMCU and the server. By intercepting and analyzing the unencrypted data, we have underscored the urgent need for stronger security measures. Moving forward to Phase 3, we will explore advanced encryption techniques, specifically focusing on implementing elliptic curve cryptography (ECC) to secure the communication channel effectively. ECC offers improved security, reduced computational overhead, and resistance against timing attacks, making it a promising solution for safeguarding the integrity and confidentiality of the transmitted data. In Phase 3, we will delve into the implementation and evaluation of ECC in the NodeMCU device, aiming to establish a robust and secure framework for IoT applications.

V. PHASE 3: IMPLEMENTING ELLIPTIC CURVE CRYPTOGRAPHY (ECC) FOR SECURE COMMUNICATION

In this phase, we will focus on implementing Elliptic Curve Cryptography (ECC) as a robust security measure for ensuring secure communication between the client-side (NodeMCU) and the server-side. ECC is a modern public-key cryptographic algorithm known for its efficiency and strong security properties. By leveraging the mathematical properties of elliptic curves, ECC offers smaller key sizes, reduced bandwidth requirements, lower energy consumption, and enhanced security compared to traditional algorithms like RSA. This makes ECC particularly well-suited for low-cost IoT devices with limited computational resources. In this phase, we will explore the principles of ECC, its advantages for IoT devices, and proceed to implement ECC on the client and server sides to establish a secure communication channel.

1. Exploring Elliptic Curve Cryptography (ECC) for IoT Devices
In this phase, we will delve into the principles of Elliptic Curve Cryptography (ECC) and its suitability for low-cost IoT devices. ECC is a public-key cryptographic algorithm that offers strong security with relatively shorter key lengths

compared to traditional algorithms such as RSA. It is based on the mathematical properties of elliptic curves and provides efficient and secure operations for key generation, encryption, and digital signatures.

ECC is particularly well-suited for low-cost IoT devices due to the following reasons:

a) *Smaller Key Sizes: ECC provides the same level of security as RSA but with significantly smaller key sizes. This advantage is crucial for IoT devices with limited computational resources, as smaller key sizes require less memory and processing power.*
b) *Reduced Bandwidth Requirements: The smaller key sizes of ECC result in shorter ciphertexts, reducing the bandwidth required for secure communication. This is beneficial for IoT devices operating on low-power and low-bandwidth networks.*
c) *Lower Energy Consumption: The computational efficiency of ECC contributes to lower energy consumption, making it suitable for battery-powered IoT devices. The reduced computational overhead results in extended battery life and increased operational efficiency.*
d) *Enhanced Security: ECC offers strong security against various cryptographic attacks, including brute-force attacks and key compromise. The mathematical properties of elliptic curves make it computationally difficult to derive the private key from the public key, ensuring the confidentiality and integrity of the communication.*

2. Client-Side implementation
In this part, we will focus on the client-side implementation of Elliptic Curve Cryptography (ECC) on the NodeMCU board. The purpose of this implementation is to establish a secure communication channel with the server using a one-sided key sharing mechanism. Since the NodeMCU will only be sending data to the server and not receiving any data in return, there is no need for a key exchange process. Instead, we will acquire the server's public key during the initial connection establishment.
   a) *Acquiring the Server's Public Key*
      During the connection establishment phase, the NodeMCU will request the server's public key. The server will respond by sending its public key to the client. This public key is necessary for the encryption process on the client-side.
   b) *Encrypting Data with the Server's Public Key*
      Once the server's public key is acquired, the NodeMCU can use it to encrypt the data that needs to be sent to the server. The encryption process involves applying the ECC algorithm, which leverages the mathematical properties of elliptic curves to ensure secure communication.
   c) *Establishing a Secure Communication Channel*
      By implementing ECC on the client-side, the NodeMCU can establish a secure communication channel with the server. The data sent from the NodeMCU will be encrypted using the server's public key, ensuring that only the server can decrypt and access the information. This provides confidentiality and integrity to the transmitted data, safeguarding it from potential eavesdroppers or unauthorized access.

   The client-side implementation of ECC on the NodeMCU board enhances the overall security of the IoT system, mitigating potential security risks and ensuring the confidentiality of the transmitted data. In the next part, we will focus on the server-side implementation of ECC to complete the secure communication channel.

3. Server-Side implementation
In this part, we will focus on the server-side implementation that handles the decrypted data received from the NodeMCU and sends it to ThingSpeak platform using ThingSpeak API.
   a) *Set up a Flask server:*
      - *Create a Flask application on the server to handle incoming requests from the NodeMCU.*
      - *Define appropriate routes to handle different types of requests.*
   b) *Receive encrypted data from the NodeMCU:*
      - *Configure the Flask server to receive encrypted data from the NodeMCU.*
      - *Define a route to handle the incoming encrypted data.*
   c) *Decrypt the received data:*
      - *Implement the decryption algorithm using the server's private key.*
      - *Decrypt the received data using the appropriate ECC decryption process.*
   d) *Extract the decrypted data:*
      - *Retrieve the decrypted data from the received ciphertext.*
      - *Extract the relevant information from the decrypted data.*
   e) *Send the extracted data to ThingSpeak:*
      - *Use the ThingSpeak API to send the extracted data to the ThingSpeak platform.*
      - *Prepare the data for sending and include any necessary metadata.*

f) *Handle server-side operations:*
   - *Implement any additional server-side operations or data processing required.*
   - *Perform any necessary validation or verification steps on the received data.*

g) *Respond to the NodeMCU:*
   - *Send a response back to the NodeMCU indicating the success or failure of the data transmission.*
   - *Provide any necessary feedback or instructions for the NodeMCU.*

h) *Repeat steps 2-7 for subsequent data transmissions:*
   - *Configure the Flask server to handle multiple requests from the NodeMCU.*
   - *Decrypt, process, and forward the data to ThingSpeak for each received ciphertext.*

Overall, Phase 3 has significantly strengthened the security and functionality of our IoT system, paving the way for reliable and secure data communication between the NodeMCU and the server. The successful implementation of ECC and the server-side infrastructure sets the stage for further advancements in our research, with the potential to extend our IoT system to broader applications and enhance its overall efficiency and security.

VI. CONCLUSION

In conclusion, this research project delved into the security aspects of IoT devices, focusing on the NodeMCU platform. Through the exploration of a vulnerability attack and the subsequent implementation of security measures, we have shed light on the importance of ensuring the confidentiality and integrity of data transmitted by IoT devices. Our findings and methodologies have been documented and are accessible for further exploration on our GitHub repository at github.com/costomato/nodemcu-mitm-attack, providing a resource for those interested in the technical details of our work.

During Phase 1, we identified a vulnerability in the NodeMCU firmware that allowed unauthorized access to sensitive information. By intercepting the data packets, an attacker could exploit this vulnerability to gain unauthorized access to the device and potentially compromise the entire IoT network. This discovery emphasized the critical need for robust security measures to protect IoT devices.

In Phase 2, we performed a Man-in-the-Middle (MITM) attack to highlight the vulnerability of NodeMCU devices and the potential risks associated with unencrypted data transmission. By intercepting and analyzing the traffic between the NodeMCU and the server, we showcased the ease with which an attacker could obtain sensitive information. This phase demonstrated the urgent need for encryption mechanisms to safeguard data during transmission.

To address these security concerns, Phase 3 focused on implementing an Elliptic Curve Cryptography (ECC) algorithm as a secure communication mechanism between the NodeMCU and the server. ECC offers several advantages, including strong encryption capabilities with relatively smaller key sizes, making it well-suited for resource-constrained IoT devices. The implementation of ECC provided secure key exchange and ensured the confidentiality and integrity of the transmitted data.

The results of this research highlight the significance of adopting robust security measures in IoT devices. By addressing vulnerabilities and implementing encryption mechanisms, we can enhance the overall security posture of IoT networks. Future research in this domain could explore additional security protocols, such as authentication mechanisms and intrusion detection systems, to further fortify IoT devices against potential threats.

In summary, this research underscores the importance of prioritizing security in IoT deployments. By understanding the vulnerabilities, implementing encryption algorithms like ECC, and adopting best practices, we can mitigate the risks associated with IoT devices and pave the way for a secure and resilient IoT ecosystem.

ACKNOWLEDGMENT

This work was supported by the invaluable guidance and support of Dr. Murthy DHR, Assistant Professor at Presidency University, Department of Computer Science and Engineering. I am grateful for his expertise in blockchain and cryptography, as well as his enthusiasm for these fields. Dr. Murthy's knowledge and teachings on Elliptic Curve Cryptography (ECC) laid the foundation for this research. The vulnerability attack on the NodeMCU, which inspired this study, was a result of an assignment given by Dr. Murthy in the course of Cyber Threats for IoT and Cloud. His mentorship and encouragement have been instrumental in the successful completion of this research project. I am deeply thankful for his contributions and the impact he has had on this work.

REFERENCES

[1] Zhang, Y., & Lee, W. (2017). Security in the Internet of Things: A Review. International Journal of Distributed Sensor Networks, 13(8), 1550147717718010.

[2] Raza, S., Wallgren, L., & Voigt, T. (2013). SVELTE: Real-time intrusion detection in the Internet of Things. Ad Hoc Networks, 11(8), 2661-2674.

[3] Alrawais, A., Alhothaily, A., Hu, C., & Cheng, X. (2017). The role of authentication in the Internet of Things. Computer Networks, 112, 137-147.

[4] Gupta, M., & Sandhu, R. (2018). The challenges of cryptographic operations in IoT devices. Network Security, 2018(2), 5-8.