Benchmarking Applicability of Cryptographic Wireless Communication over Arduino Platforms

Carolina Vázquez Torres
Northeastern Illinois University, c-vazquez11@neiu.edu

Follow this and additional works at: https://neiudc.neiu.edu/uhp-projects
Part of the Computer Sciences Commons

Recommended Citation
https://neiudc.neiu.edu/uhp-projects/3
HONORS SENIOR PROJECT
ACCEPTANCE AND APPROVAL FORM

Carolina Vázquez Torres

Benchmarking Applicability of Cryptographic Wireless Communication over Arduino Platforms

This thesis has been reviewed by the faculty of the NEIU Honors Program and is found to be in good order in content, style, and mechanical accuracy. It is accepted in partial fulfillment of the requirements of the NEIU Honors Program and graduation with honors.

Ahmed E. Khaled, Computer Science Department
Faculty Advisor

Graciela Perera, Computer Science Department
Faculty Reader

Nadja Insel, Earth Science Department
Honors Curriculum & Standards Board

Prof. Jon B. Hageman, Department of Anthropology
Coordinator, University Honors Program
ABSTRACT

The spaces around us are becoming equipped with devices and appliances that collect data from their surroundings and react accordingly to provide smarter networks where they are interconnected and able to communicate with one another. These smart networks of devices and appliances along with the applications that utilize them build smart spaces known as Internet of Things (IoT). With the on growing popularity of such smart devices (e.g., smart cars, watches, home-security systems) and IoT, the need for securing these environments increases. The smart devices around us can collect private and personal information, and the challenge lies in maintaining the confidentiality of the collected data and preventing unsecured actions—from tapping into surveillance cameras to tracking someone’s daily schedule. For example, digital health, devices that record personal data from blood pressure, heart rate, weight and daily activities sensors are storing the personal data of users for processing and monitoring and may give future recommendations. If such personal information reaches unwanted third parties who distribute or use the data without user consent or knowledge, they are attacking the user’s confidentiality. Therefore, selecting the appropriate security protocols and procedures is critical. The limited processing, storage and power capabilities. In this thesis, the focus is to provide an experimental benchmark study that shows the cost (e.g., processing time of encryption and decryption algorithms) of applying different security protocols on restricted devices equipped with lightweight Bluetooth or Wi-Fi communication modules over the Arduino Uno sensor platform.
TABLE OF CONTENTS

ABSTRACT ................................................................................................................................. iii

LIST OF TABLES ......................................................................................................................... vi

LIST OF FIGURES ...................................................................................................................... vii

CHAPTER 1: INTRODUCTION .................................................................................................... 1

CHAPTER 2: REVIEW OF LITERATURE ..................................................................................... 6

2.1. INTERNET OF THINGS AND SECURITY SHORTCOMINGS .............................................. 6

2.2. A REVIEW ON STATE OF ART ON SECURITY IN IOT ..................................................... 8

CHAPTER 3: METHODOLOGY ................................................................................................... 11

3.1. FUNCTION 1: BUILDING BLUETOOTH AND WI-FI NETWORKS ................................. 11

3.2. FUNCTION 2: CREATING AND DERIVING SYMMETRIC AND ASYMMETRIC ALGORITHMS .................................................................................................................. 13

3.3. FUNCTION 3: COMBINING FUNCTION 1 AND FUNCTION 2 ......................................... 18

CHAPTER 4: BENCHMARKS ...................................................................................................... 19

4.1. TECHNICAL SPECIFICATIONS ......................................................................................... 20

4.1.1. NETWORK TOPOLOGIES ............................................................................................. 20

4.1.2. WI-FI ENVIRONMENTS FOR TESTING ...................................................................... 24

4.1.3. BLUETOOTH ENVIRONMENTS FOR TESTING .......................................................... 26

4.2. TESTS PERFORMED ON EACH CRYPTOGRAPHIC ALGORITHM ............................... 29
TEST 1: AVERAGE TIME IT TAKES TO ENCRYPT AND DECRYPT MESSAGES WITH CAESAR CIPHER, AES AND ED25519 .......................................................... 29

TEST 1.1. ENCRYPTING AND DECRYPTING MESSAGES WITH CAESAR CIPHER ........................................................................................................... 29

TEST 1.2. ENCRYPTING AND DECRYPTING MESSAGES WITH AES ............... 30

TEST 1.3. AUTHENTICATING MESSAGES WITH ED25519 .................................. 32

TEST 2: AVERAGE TIME IT TAKES TO SEND A SIMPLE MESSAGE OVER WIRELESS PROTOCOLS ................................................................................ 32

TEST 2.1. ROUND TRIP BENCHMARK FOR WI-FI ENVIRONMENT .................. 33

TEST 2.2. ROUND TRIP BENCHMARK FOR BLUETOOTH ENVIRONMENT ...... 33

TEST 3: ROUND TRIP BENCHMARK FOR WIRELESS ENVIRONMENTS USING CAESAR CIPHER AND AES ................................................................. 34

TEST 3.1. WI-FI ENVIRONMENT WITH CAESAR CIPHER VS AES AND MICROPHONE SENSOR ...................................................................................... 36

TEST 3.2. WI-FI ENVIRONMENT WITH CAESAR CIPHER VS AES AND PHOTOSENSOR .......................................................................................... 37

CHAPTER 5: CONCLUSION ................................................................................. 39

REFERENCES ..................................................................................................... 42
LIST OF TABLES

TABLE 4.1. COMPARATIVE SPECIFICATIONS OF BLUETOOTH AND WI-FI PROTOCOLS ................................................................. 24

TABLE 4.2. ESP8266 WI-FI MODULE AND HC-05 BLUETOOTH MODULE SPECIFICATIONS ........................................................................... 28

TABLE 4.3. CAESAR CIPHER AVERAGE ENCRYPTION TIMES ............................................. 29

TABLE 4.4. CAESAR CIPHER AVERAGE DECRYPTION TIMES ............................................. 30

TABLE 4.5. AES AVERAGE ENCRYPTION TIMES ................................................................. 31

TABLE 4.6. AES AVERAGE DECRYPTION TIMES ................................................................. 31

TABLE 4.7 ED25519 AVERAGE VERIFICATION TIME ......................................................... 32

TABLE 4.8. CLIENT/SERVER AVERAGE COMMUNICATION TIME (WI-FI) ............. 33

TABLE 4.9. CLIENT/SERVER AVERAGE COMMUNICATION TIME (BLUETOOTH) ................................................................. 34

TABLE 4.10. ROUND TRIP BENCHMARK IN WI-FI CLIENT/SERVER ENVIRONMENT (MICROPHONE SENSOR) ................................................................. 37

TABLE 4.11. ROUND TRIP BENCHMARK IN WI-FI CLIENT/SERVER ENVIRONMENT (PHOTORESISTOR) ................................................................. 38
LIST OF FIGURES

FIGURE 1.1. ARDUINO UNO BOARD ........................................................................................................ 3

FIGURE 3.1. CLIENT-SERVER BLUETOOTH NETWORK ......................................................... 12

FIGURE 3.2. CLIENT-SERVER WI-FI NETWORK ........................................................................... 12

FIGURE 4.1. TEST 1: ENCRYPTION AND DECRYPTION OVER SINGLE
ARDUINO (NO CLIENT-SERVER) ........................................................................................................... 21

FIGURE 4.2. TEST 2: CLIENT AND SERVER WIRELESS COMMUNICATION (NO
ENCRYPTION/DECRYPTION) ........................................................................................................... 22

FIGURE 4.3. TEST 3: CLIENT AND SERVER WIRELESS COMMUNICATION
WITH ENCRYPTION/DECRYPTION ................................................................................................... 23

FIGURE 4.4. COMPONENTS OF WI-FI ENVIRONMENT FOR TEST 2 ............... 25

FIGURE 4.5. ESP8266 MODULE (A: MODULE, B: PINOUT) ............................................... 25

FIGURE 4.6. COMPONENTS OF WI-FI ENVIRONMENT FOR TEST 3 ............... 26

FIGURE 4.7. COMPONENTS OF BLUETOOTH ENVIRONMENT FOR TEST 2 ..... 27

FIGURE 4.8. HC-05 MODULE (A: MODULE, B: PINOUT) ................................................. 27

FIGURE 4.9. COMPONENTS OF BLUETOOTH ENVIRONMENT FOR TEST 3 ..... 28

FIGURE 4.10. COMPONENTS OF MICROPHONE SENSOR LM393 (A)
PHOTORESISTOR (B) .......................................................................................................................... 35

FIGURE 4.11. HARDWARE COMPONENTS FOR TEST 3, SECTION I OF THE WI-
FI ENVIRONMENT .............................................................................................................................. 35
FIGURE 4.12. HARDWARE COMPONENTS FOR TEST 3, SECTION II OF THE WI-FI ENVIRONMENT
CHAPTER 1: INTRODUCTION

Current technologies have allowed the proliferation and spread of ubiquitous computing in users’ spaces. These devices (e.g., smart thermostats, smart watches) collect data (e.g., a room’s temperature, heartrate) from their environment (e.g., smart-homes) and process the data accordingly, creating smart networks between themselves and the applications that manage them and thus, building smart spaces known as Internet of things (IoT) (Jones, 2018). The devices that make up IoT have low processing capabilities and memory storage units, and communicate using different protocols and modules (e.g., Bluetooth, Wi-Fi, Ethernet). IoT has different scales: personal IoT (e.g., smart appliances, watches, cars, lamps, etc.), industrial IoT (e.g., smart factory floors with security sirens and machinery) and at-scale IoT (e.g., a larger side to IoT that includes smart transportation vehicles, traffic lights and street lamps). These scales mainly differ in the number of devices and sensors they each have, affecting the management and type of applications used. All the scales of IoT engage multiple devices in one way or another and allow interconnectivity through different communication protocols. Take for example a wireless personal area network (WPAN) in an office scenario (an example of personal IoT) where a user connects their laptop to a printer, a mouse, and/or coffee machine through Wi-Fi, or another module where a user connects their smartphone to their smartwatch through Bluetooth. In this thesis, we focus on the personal IoT.

The creation of such interactive environments raises security concerns as confidential data being transferred from a device to another device needs to be protected from malicious actuators (e.g., credit card thieves who take advantage of a user doing an online purchase through a public Wi-Fi network). Security threats on IoT can range from
small scale attacks (personal data hijacked from personal devices) to larger scale attacks (industrial IoT threats on nuclear powerplants or train services that can affect an organization financially). Wan and Liu (2012) explain that the IoT faces challenges because the traditional internet provides a pathway to the extension of the internet where all the devices will be interconnected and create communication channels through IoT. For instance, a study on 50 smart home devices and their security mechanisms indicated that smart devices are vulnerable to common attacks (i.e. brute force methods on default credentials and denial of service (DoS) attacks) (Barcena and Wueest, 2015). In addition, Barcena and Wueest found that none of the devices tested, within their study, implemented strong passwords or used mutual authentication. When two devices ensure that the message they received is from an authorized sender, they minimize any potential risk of receiving malicious data. Therefore, the absence of security within the communication of interconnected devices increases the possibility of a bigger scale attack on the network.

Arduino Uno (shown in Figure 1.1.) is a microcontroller-based sensor platform that can send and receive information to/from other devices (e.g., data collected by sensors, messages, requests), allow different sensors to be plugged with the use of a circuit board and programmed with software in C++ coding language (Badamasi, 2014). A microcontroller (MCU) executes a single program or specific task tailored specifically to the input and output relationship of the applications and devices. Based on the input received, microcontrollers produce a specific output, for example a connected coffeemaker only needs to perform simple routines such as brewing coffee every morning at a specific hour. A microprocessor (MPU) on the other hand, has greater processing
capabilities, functionalities and features. For example, smartphones have an MPU that handles multiple tasks simultaneously or in parallel. Given that the Arduino Uno is an MCU, a user can create a program that reads the data from a temperature sensor connected to the Arduino to sense the room’s current temperature and then send the data back to another device or application. This program is then uploaded to the microcontroller and it executes the uploaded program by the uploader. One of the ways in which the IoT’s security can be tested is through the implementation of an Arduino Uno network. An Arduino Uno board is a platform upon which software developers can test the security measurements of current and future applications. The environment created through the connection of Arduinos and sensors creates potential IoT devices. In this sense, large-scale attacks within infrastructure, health, transportation and public sectors can be prevented by testing the security of the communication channels of these devices.

Figure 1.1. Arduino Uno Board (photo by: Arduino Uno R3)
We built a wireless network, using Bluetooth and Wi-Fi, of Arduino boards, and then tested two security protocols to securely send and receive data within the network. One of the security protocols we will use is the Advanced Encryption Standard (AES) symmetric encryption algorithm. Symmetric encryption uses a single secret key (of 128, 192 or 256 bits) to encrypt and decrypt data (Singh and Supriya, 2013). Caesar Cipher is a substitution cipher that shifts each letter in the plaintext message a certain number of spaces down or up the given alphabet. For example, if the user sets the shift to be 2, the letter “A” within the plaintext will be encrypted as “C” in the cipher, “B” would become “D” and so on. The third security protocol is the ED25519, an asymmetric encryption algorithm. Asymmetric encryption or public key encryption uses two keys: a private and a public key. The public key is used to encrypt the data and the private key to decrypt (or vice versa) (Singh and Supriya, 2013). The ED25519 algorithm is a digital signature based on the elliptic curve modulo $2^{255} - 19$. The advantage of using symmetric key encryption algorithms (e.g., AES and Caesar Cipher) is the low storage and processing requirements used to store the key. Arduinos have low memory capabilities, therefore, having the least amount of memory use will work towards it's advantage. On the other hand, the lack of more security layers increases its vulnerability to attacks. Asymmetric key encryption (e.g., Elliptical Curve, Ed25519) has the advantage of a private key that only the user has knowledge of and uses it to decrypt the data it receives along with a public key that is shared with the rest of the devices in the network. The use of the two keys decreases the possibility of acquiring access to the data being transferred without accessing the appropriate keys. The disadvantage, however, is the amount of memory
required to save the keys. For instance, the Arduino will have to save its public and private key in addition to the public keys of the devices it's communicating with.

The purpose of the thesis is to secure the communication and data transferred between IoT devices (e.g., Arduino -sensor platform) by building a network of two Arduinos, Caesar cipher as a base case, symmetric (AES) and asymmetric encryption (ECC) algorithms. The network of Arduinos is composed of a server and a client: the server holds sensors (photoresistor and sound sensor) and reports data based on the requests sent by the client. Through the creation of an Arduino sketch and java program the networks will communicate through Bluetooth and WiFi modules. The thesis implements different cryptographic measures to benchmark message secrecy on Arduino Uno communication over Bluetooth and Wi-Fi. This paper is organized into four main chapters, in addition to this one. Chapter 2 identifies past research on the cryptography algorithms and vulnerabilities of IoT. Chapter 3 focuses on the methodology of the thesis. In Chapter 4 you will find the benchmarks collected in the duration of the research. Chapter 5 provides conclusion and possible solutions to the deficiencies of the security algorithms implemented in IoT environments.
CHAPTER 2: REVIEW OF LITERATURE

Technology has served as a pathway for user comfortability and an easier lifestyle for the consumer. One of these platforms is the Internet of Things (IoT) or smart spaces equipped with all sorts of sensors (e.g., sensors with the capabilities to sense temperature, sound, smoke, and fire). The IoT scope ranges from personal uses (e.g., smart watches, coffeemakers, baby monitors, cars) to industrial applications. Researchers in 2012 expected that IoT devices would be found in the medical field, intelligent transportation, manufacturing, public sector, and other infrastructures (Jones, 2018; Suo, Wan, Zou, and Liu, 2012). IoT has gained popularity over the years, and it is expected to continue to grow exponentially (Nagamalla, Vishwesh and Varanasi, 2017; Suo, Wan, Zou, and Liu, 2012). The application of IoT has the benefit of low prices (as devices can be cheaper) and high potency, giving it more popularity (K.V.S.S.S.S, Gunasekaran and Reddy, 2002; Chang, 2014).

2.1. Internet of Things and Security Shortcomings

The IoT can be divided into four major categories, each with specific functions. The four layers of the IoT are (1) physical layer, (2) network layer (or edge), (3) cloud layer, and (4) application layer. Layer one, the physical layer is composed of physical devices and objects such as televisions, toasters, and watches. These devices use different communication technologies and protocols to be able to communicate with one another or the rest of the IoT layers. The second layer, the network layer, collects the raw data provided by the devices in the physical layer and translates it into a format the rest of the upper layers can process. The third layer, the cloud layer provides services on the internet after it processes and analyses the data. In addition, it manages devices and data and acts
as a bridge between software applications and layer one devices. Finally, the application layer develops software that uses information provided by layer three with the end goal of accessing layer one devices and data. Given the amount of data and processes that occur at each layer, they are all vulnerable to different security threats, and the implementation of secure practices are needed more than ever.

IoT devices face security problems that lead to security implementations of authentication, integrity and confidentiality. Authentication is the process of ensuring that the user requesting/sending data is who they say they are. Integrity is verifying that the data received has not been tampered by unwanted users. Lastly, confidentiality is maintaining the data being sent/received private, or only shared with those who have permission. These three security requirements within smart spaces can be implemented using two main types of procedures: symmetric and asymmetric cryptographic algorithms (Jones, 2018; Suo, Wan, Zou and Liu, 2012). Symmetric cryptography uses a shared secret key to decrypt and encrypt the data (e.g., Caesar Cipher, AES) and asymmetric cryptography uses a pair of keys (private and public) to encrypt and decrypt data (e.g., RSA, DSA). Through the application of symmetric cryptography, the devices share one private key that is used to transform the plaintext into ciphertext through encryption and the ciphertext into plaintext through decryption. On the other hand, asymmetric cryptography applies the use of a pair of keys: public and private. The public key is used to transform plaintext into ciphertext through encryption and the private key is used to decrypt and find the readable plaintext. The implementation of either cryptographic algorithm decreases the likelihood of tapping into the communication channels of these devices. However, there still exists a security gap among these smart spaces.
2.2. A Review on State of Art on Security in IoT

Jones (2018) describes the vulnerabilities within the IoT attack surface. Specifically, he highlights historic attacks that demonstrate that old security exploits from an x86 machine can access common IoT devices. Thus, the lack of proper implementation of security practices by manufacturers is demonstrated by the alarming 70% of IoT devices that do not support security principles (e.g., granular access control, strong encryption). Jones emphasizes on the importance of sufficient encryption in regards to user data, private keys and public keys. He suggests the enforcement of security as early as the design stage of the IoT, from allowing users to change default passwords from the beginning to security level at the transistor level. His proposed recommendations include web-based solutions (e.g., proper testing of web interfaces), ethical implementation of strict codes and guidelines within organizations, and training employees on how to properly use an IoT device.

Nagamalla and Varanas (2017) provided a summary of four security measures using sources that varied from application layer protection to key and certificate management to ICT security standards. The four security measures that were exploited in the systematic review were Cisco, Floodgate, Constrained Application Protocol (CoAP) and OSCAR (object-oriented security framework for IoT). The authors concluded that while the frameworks had different advantages based on their IoT connected comparative criteria (Policies and Processes Adaptation, Management and Audit, Service Level Agreement/Security Risk Assessment and Applicability), they also had criterion that was insufficiently developed. For instance, while CoAP was best fit in application security framework and provided easy mapping to HTTP at the gateway, it does not fulfill the
audit and control, and service level agreement criterion when compared to the other three security frameworks. In addition, the Floodgate and OSCAR security frameworks fulfilled all of the criterion used by the researchers.

According to Suo, Wan, Zou and Liu (2012) IoT has higher security needs in availability and dependability given that these devices are used in important sectors of the national economy. For example, the physical layer (or perceptual layer in their article) has difficulty in setting security protection systems due the low power and storage capacities embedded in the IoT devices. They suggest the need of lightweight data encryption, increasing confidentiality between nodes and preventing illegal node access while overpassing the low quantity of resources available directly from the devices. Furthermore, the authors suggest by-hop encryption when the security requirements of an organization are not too high and end-to-end encryption when these organizations are looking for high-security implementations.

Due to the low power capabilities and short-comings of the devices, lightweight communication protocols can be used to connect the devices: Bluetooth and Wi-Fi. Bluetooth and Wi-Fi are both wireless Institute of Electrical and Electronics Engineers (IEEE) protocols. Bluetooth’s architecture consists of a piconet structure that holds up to eight devices: one device that acts as the master (controls communication between other devices and itself) and up to seven salve devices that serve the master (Lee, Su, and Shen, 2007; Scarfone and Padgette, 2008). Bluetooth provides a wireless connectivity to various devices and has vulnerabilities “inherited in the protocol” (Hager and Midkiff, 2003) or caused by the implementation. On the other hand, Wi-Fi allows up to 2007 devices to browse the internet when they are connected to an access point through a basic
service set (Lee, Su, and Shen, 2007). The comparative study of Lee, et al. (2007), found that for data coding efficiency, Bluetooth was the best solution for small and large data sizes while Wi-Fi was only preferred for large data sizes. In addition, because Bluetooth consumes less power compared to Wi-Fi, it was more suitable for low data rate applications within low-power devices. An example of low-power devices is Arduino Uno boards: hardware devices that are used to test software for devices within IoT.

Arduino Uno is a microcontroller that can receive and send information to other devices. The Arduino is connected to different sensors with the use of a circuit board and programmed with software in C++ coding language (Badamasi, 2014). For example, a user can create a program that requires a temperature sensor connected to the circuit board to send the room’s temperature. This program is then uploaded to the microcontroller and it executes the specific program sent. The board is a platform upon which software developers can test the security measurements of current and future applications. Arduino Uno gives the possibility of collecting benchmarks of the different capabilities of Bluetooth and Wi-Fi. This window of opportunity can then be utilized by application makers to decide which protocol suits their application best, giving them the pros and cons of implementing each communication channel.

In the next chapter we will discuss the methodology of the thesis. We describe the process of assigning different tests depending on the cryptographic algorithm implemented in each step of the thesis. In addition, we review the code created for every phase of the thesis.
CHAPTER 3: METHODOLOGY

In this chapter, we will represent our work to apply different security protocols (symmetric and asymmetric) on a constrained IoT device: Arduino Uno. The project has three main functions: 1) creating the wireless networks, 2) implementing symmetric and asymmetric cryptographic algorithms and 3) combining aspects one and two into a single function where each wireless protocol implements the asymmetric and symmetric cryptographic algorithms. In the first function, we built the wireless systems for the Arduino devices (client - server) to communicate. In the second function we secured the communication between client and server (e.g., the client requesting the room’s temperature from the sensors attached to the server) with the Caesar Cipher and AES algorithms. In function three we secured the overall wireless network communication between client and server created in function one by applying the cryptographic measures from function two.

3.1. Function 1: Building Bluetooth and Wi-Fi Networks

In the first function, two networks are created: Bluetooth and Wi-Fi networks. The Bluetooth network consists of two Arduino boards: one acting as server and the other as client (Figure 3.1.). The Wi-Fi network consists of two Arduino boards: one server and the other client (Figure 3.2). Each environment also has sensors (photoresistor and sound detection sensor) and a communication module (HC-05 for Bluetooth and ESP8266 for Wi-Fi; described in Chapter 4). The sensors and communication modules are connected to the Arduino through breadboard and circuit jumper wires. Once the physical configuration is accomplished, software programs (as illustrated in pseudo codes 1, 2 and 3) are created and uploaded to the client/server Arduinos accordingly. The Arduino sketches allowed multiple Arduinos to send and receive requests and encrypt/decrypt
different messages received by using asymmetric and symmetric algorithms. The client encrypts the request and the server side decrypts the request to later build an encrypted response to send back to the client who decrypts it to see the results.

Figure 3.1. Client-Server Bluetooth Network

Figure 3.2. Client-Server Wi-Fi Network
3.2. Function 2: Creating and Deriving Symmetric and Asymmetric Algorithms

The second function focuses on cryptographic measures to secure the data. Each network (Bluetooth and Wi-Fi) implements the same symmetric and asymmetric algorithms to benchmark message secrecy on the different types of light-weight networks. The two primary algorithms the thesis focuses on are Caesar Cipher and AES, however, for this portion of the thesis benchmarks are collected for the Elliptical Curve (ED25519). Given that the Caesar Cipher does not require profuse memory allocation, the thesis uses the cipher to establish a base case within the low-memory devices (Arduino Unos). The Caesar Cipher, an affine cipher, is also used to encrypt and decrypt the communication between client and server. An affine cipher is a monoalphabetic substitution cipher where each letter in the given alphabet has a corresponding numeric equivalent, which is used to encrypt and decrypt using a mathematical function. The Caesar Cipher, also known as a shift cipher, encrypts and decrypts the original message based on the shift (whether it’s up or down the alphabet). For this cipher, a master key is set as an integer value of thirteen to encrypt and decrypt the message. For encryption, the master key is added to each character of the message (as illustrated in Pseudo Code 1). To attain the original message, the decryption method takes the message (in cipher version), size of the message and master key and subtracts the master key from each character (as illustrated in Pseudo Code 2).

On the other hand, AES is used as a more dynamic communication channel for a client-server environment. The private key used by the Arduinos when communicating through AES is pre-shared and saved in each device’s memory. We used AES-128, which requires a 128-bit master key, or in other words, a 24-character key (in base64
This key is then used to both encrypt and decrypt all communication between the client and server Arduino Unos within the same environment either Bluetooth or Wi-Fi (as illustrated in Pseudo Code 3 and 4).

**Pseudo Code 3.1. Caesar Cipher Encryption**

**Description:** This method receives the plaintext, size of the plaintext and the master key that is shared with both client and server Arduino Unos in the networks. For example, if the plaintext sent is “hello”, the size of plaintext received is five, which is then used to iterate the same number of times and create the cipher by shifting each character in the message with the value of the master key.

**Procedure** ENCRYPT METHOD (plaintext, size of plaintext, master key)

```plaintext
for i = 0 to i < size of plaintext do
    plaintext[i] = plaintext[i] + master key;
end for
store plaintext as InputString
end procedure
```

**Pseudo Code 3.2. Caesar Cipher Decryption**

**Description:** The decryption method takes the cipher created by the encryption method, the size of the cipher and master key. It then proceeds to enter a loop that iterates the same amount of times as the size of the cipher in order to subtract the master key from each character and conclude with the original plaintext message/request.

**Procedure** DECRYPT METHOD (cipher, size of cipher, master key)

```plaintext
for int i = 0 to i < size of cipher do
    cipher[i] = cipher[i] - master key;
end for
store cipher as InputString
end of procedure
```
Pseudo Code 3.3. AES Encryption at the Client Side

**Description:** The AES Encryption method takes the plaintext version of the message or request, length of the plaintext, a cipher variable as placeholder to store the cipher version of the plaintext, the master key, 128 (size of the master key), and an initial vector. The initial vector is an arbitrary number (provided by the AES library) that is used along the master key to prevent repetition in data encryption. Where the `aes.do_aes_encrypt` (function in AES-Master Library by Brian Gladman) encrypts data one 128-bit block at a time by XORing each block with a key, and then performing substitution and permutation functions.

**Procedure** LOOP

```plaintext
    if client and server are connected then
        Store message in request from client
        Call aes.do_aes_encrypt(request, length of plain, cipher, master key, 128, initial vector)
        client print cipher
    end if
    //stop waiting for input from server
end procedure
```
Pseudo Code 3.4. AES Decryption at the Server Side

**Description:** The AES Decryption method takes the encrypted message and stores it in the *plain* variable, takes the length of the plain, *check* (place holder that will store the message once decrypted), the master key, 128 for the bit size of the key, and the initial vector (same initial vector as the AES Encryption method. For this portion, the same library, AES-master is used to decrypt the communication and the *aes.do_aes_decrypt()* method also decrypts the data by blocks and a combination of mathematical functions.

**Procedure** LOOP

```
if client and server are connected then
    Store incoming message in *request*
    Cast *request* as a byte array and store in *plain* array
    Call *aes.do_aes_decrypt(request, length of plain, check, master key, 128, initial vector)*
    Store *check* in *decrypted*
    Read *decrypted* and answer accordingly
end if
// return the answer to the blocked requester
```

end procedure
**Pseudo Code 3.5. ED25519 at the Client side**

**Description:** For the ED25519 on the client side, a private key of 32 bits is randomly generated and the public key (of 32 bits) is derived from that same private key. The message the client wants to send is then signed using the client’s private and public key to send it to the server. For this algorithm, we used the ED25519 sub library (in _Crypto-Master_ library by Rhys Weatherly).

**Procedure** LOOP

- Store `privateKey` and `publicKey`
- Create `message`
- Call `sign(signature, privateKey, publicKey, message, 12);`
- Send the signed message to the server

**end procedure**

---

**Pseudo Code 3.6. ED25519 at the Server side**

**Description:** For the ED25519 on the server side, a private key of 32 bits is randomly generated and the public key (of 32 bits) is derived from that same private key. The server however, also stores the client’s public key. The client’s public key is used to authenticate the message the server received. For this algorithm, we used the ED25519 sub library (in _Crypto-Master_ library by Rhys Weatherly).

**Procedure** LOOP

- Store `privateKey` and `publicKey`
- Store the message received
- Call `verify(signature, publicKey, message, 12)`

**end procedure**
3.3. Function 3: Combining Function 1 and Function 2

The previous two functions are merged into a single system. Each wireless network now has a connectivity protocol (Bluetooth or Wi-Fi), encryption algorithms (Caesar Cipher, AES and ECC) and sensors that collect data from the environment. For example, if the client communicates with the server through the Wi-Fi module and wants the room's temperature, the client encrypts the request (e.g., using Caesar Cipher), sends it via Wi-Fi and waits for a response. On the server side, the device receives the request, decrypts it (e.g., using Caesar Cipher), extracts the data accordingly by reading from the temperature sensor, encrypts it and sends it back to the client. This logic is followed by both wireless networks (Bluetooth and Wi-Fi), except they each respond accordingly depending on the cryptographic algorithm being used for that communication session and the data the client is requesting.

In the next chapter, we will present the benchmark collected on running the different security protocols on the Arduino Unos over the two wireless networks. We present figures of how each connection is built depending on the environment being tested, and tables of the data collected in a set of three tests. Each test implements the functions presented in this chapter. Test 1, tests the functionality of the wireless networks created in Function 1, Test 2 measures the speed of the cryptographic algorithms created or derived in Function 2 in a client-server session, and Test 3 measures the speed at which each wireless networks encrypt and decrypt requests from the client and respond with the data wanted.
CHAPTER 4: BENCHMARKS

This chapter provides insight into the various benchmarks collected in three tests. Test 1 measures the average time it takes each cryptographic algorithm we mentioned in Chapter 3 to encrypt and decrypt a message on the single Arduino Uno board without a wireless network communication (see Figure 4.1. for this test’s session diagram). Test 2 measures the roundtrip time it takes the client and server for a single session to occur (see Figure 4.2. for this test’s session diagram). In this test, the client and server communicate through a wireless communication network (Bluetooth or Wi-Fi) but do not implement any cryptographic algorithms. Test 3 combines Tests 1 and 2 into a single test: measures the average time it takes an Arduino client in a wireless environment (Bluetooth and Wi-Fi) to encrypt request, send it to the server and process the server’s encrypted response using both Caesar Cipher and AES cryptographic algorithms (see Figure 4.3. for this test’s session diagram).

The benchmarks collected from all three tests derived different information. Test 1 shows that the variation between key size and message size had no considerable effect on the total time it takes to encrypt and decrypt the message regardless of the algorithm used (AES or Caesar Cipher). Test 2 reveals that under a Bluetooth environment, the message is sent and received at a faster pace than under a Wi-Fi environment. Due to a time constraint the Bluetooth environment for Test 3 is not implemented. However, the results of Test 3 under the Wi-Fi environment show that if the server had a microphone sensor or a photoresistor there is no major difference between the results if the same algorithm is used. The difference in results is between the Caesar Cipher and AES, it
takes AES a longer period of milliseconds to encrypt and decrypt the data without the
type of sensor having a major effect.

4.1. Technical Specifications

4.1.1. Network Topologies

Two different environments are created for the Arduino devices to communicate
the data collected by the sensors— one environment used Bluetooth, IEEE 802.15.1
protocol, and the other Wireless Fidelity (Wi-Fi)/ IEEE 802.11. Bluetooth implements a
wireless radio system used in wireless personal area networks (WPAN). Bluetooth has
two connectivity topologies and the thesis employs the piconet topology to connect the
master Arduino Uno (client) to the slave Arduino Uno (server). A piconet topology is a
wireless personal area network (WPAN) created by a master device and up to eight
additional devices that serve as slaves. These slaves use a point-to-point communication
mechanism to communication with the master and the master can use point-to-point or
point-to-multipoint to communicate with the slaves. Wi-Fi on the other hand, includes
standards used for wireless local area networks (WLAN). This protocol has a basic cell
called a basic service set (BSS) comprised of fixed stations in which devices can only
directly communicate with one another under their BSS. For example, a smart home with
devices like temperature sensors, security alarms, and smoke sensors only has the
capacity of intercommunication between the devices because they have the same access
point (a router for example). For more comparative details on both standards please refer
to Table 1, derived from Lee, Su and Shen (2007) comparative study of wireless
protocols.
Figure 4.1. Test 1: Encryption and Decryption over Single Arduino (no Client-Server)
Figure 4.2. Test 2: Client and Server Wireless Communication (no Encryption/Decryption)
Figure 4.3. Test 3: Client and Server Wireless Communication with Encryption/Decryption
Table 4.1. Comparative specifications of Bluetooth and Wi-Fi protocols

**Description:** This table provides a summary of the specifications of each wireless protocol used for Tests 2 and 3.

<table>
<thead>
<tr>
<th>Standard</th>
<th>Bluetooth</th>
<th>Wi-Fi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency Band</td>
<td>2.4 GHz</td>
<td>2.4 GHz, 5GHz</td>
</tr>
<tr>
<td>Nominal TX Power</td>
<td>0 -10 dBm</td>
<td>15-20 dBm</td>
</tr>
<tr>
<td>Number of RF Channels</td>
<td>79</td>
<td>14</td>
</tr>
<tr>
<td>Spreading</td>
<td>Frequency-hopping spread spectrum (FHSS)</td>
<td>Direct Sequence Spread Spectrum (DSSS), Complementary Code Keying (CCK), OFDM</td>
</tr>
<tr>
<td>Basic Cell</td>
<td>Piconet</td>
<td>Basic Service Set (BSS)</td>
</tr>
<tr>
<td>Authentication</td>
<td>Shared Secret</td>
<td>WPA2(802.11i)</td>
</tr>
<tr>
<td>Data Protection (Cyclic Redundancy Check-CRC)</td>
<td>16-bit CRC</td>
<td>32-bit CRC</td>
</tr>
</tbody>
</table>


### 4.1.2. Wi-Fi Environments for Testing

The Wi-Fi environments varied depending on the test being performed. Test 2 measures the average time it takes the two devices to send and receive a plain message, in other words, we measure the total round trip time it takes the plain request from the client Arduino to be received by the server Arduino as well as the plain response from the server back to the client. Therefore, this environment (see Figure 4.4. for Wi-Fi environment) only required two Arduino boards and the ESP8266 Modules (see Figure 4.5. and Table 4.2 for ESP8266 Module specifications), which provides Wi-Fi capabilities. Test 3 is based on Test 2 and adds a photoresistor and sound detection sensor (see Figure 4.6.). In addition, Test 3 also implements symmetric encryption using the
Caesar Cipher and AES algorithms. The client creates the request and encrypts it using Caesar Cipher or AES (depending on the algorithm being used), and the server decrypts it using the same cryptographic algorithm to later send the encrypted version of the data collected by the sensors attached.

Figure 4.4. Components of Wi-Fi Environment for Test 2

Figure 4.5. ESP8266 Module (a: Module, b: Pinout) (Photo by: ESP8266 Features)
4.1.3. Bluetooth Environments for Testing

The Bluetooth environments varied depending on the test being performed, similar to the Wi-Fi environment. Test 2 for the Bluetooth Environment also measures the round-trip time it takes the client to send a plain message to the server to respond back to the client. The Bluetooth environment for Test 2 (as illustrated in Figure 4.7.) only required an HC-05 Module (see Figure 4.8. and Table 4.2. for HC-05Module specifications) per Arduino Uno. Test 3 for the Bluetooth Environment, like Test 3 of the Wi-Fi environment, built from Test 2 by adding a photoresistor sensor and sound detection sensor to the server in the Bluetooth environment created through the HC-05 modules (as illustrated in Figure 4.9.). Additionally, Test 3 implements the Caesar Cipher and AES algorithms to encrypt and decrypt the communication between client and server. The client sends an encrypted request using Caesar Cipher and AES and the server decrypts the request using the same algorithm, encrypts the data requested and sends it back to the client.
Figure 4.7. Components of Bluetooth Environment for Test 2

Figure 4.8. HC-05 Module (a: Module, b: Pinout) (Photo by: HC-05 Specifications)
Table 4.2. ESP8266 Wi-Fi module and HC-05 Bluetooth module specifications

Description: This table has hardware specifications for the two wireless communication protocols (Wi-Fi and Bluetooth).

<table>
<thead>
<tr>
<th>Specifications</th>
<th>ESP8266 Module</th>
<th>HC-05 Bluetooth Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Supply</td>
<td>+3.3V</td>
<td>+5V</td>
</tr>
<tr>
<td>Protocol</td>
<td>TCP/IP protocol stack</td>
<td>Bluetooth Specification v2.0+EDR</td>
</tr>
<tr>
<td>Range</td>
<td>360 m</td>
<td>&gt;100m</td>
</tr>
<tr>
<td>Other</td>
<td>1MB Flash Memory</td>
<td>Can operate in Slave (default mode), Master and Master/Slave modes</td>
</tr>
</tbody>
</table>

Sources: ESP8266 Features, HC-05 Specifications, Rhydo Team
4.2. Tests Performed on each Cryptographic Algorithm

Test 1: Average time it takes to encrypt and decrypt messages with Caesar Cipher, AES and ED25519

Test 1 measures the average time it takes each cryptographic algorithm (Caesar Cipher, AES and ED25519) to encrypt and decrypt a message in a single Arduino without any network communication. For this test, three messages of different sizes are created to test the effect of the message size on the speed at which the messages are encrypted and decrypted. To avoid outlier data, we are using the average time, in milliseconds, it takes for the messages to be encrypted and decrypted within a loop that iterates 1000 times.

Test 1.1. Encrypting and Decrypting Messages with Caesar Cipher

To test the Caesar Cipher, we created three different master key variants. Each key variant represents the number of times each character within the plaintext version of message must be shifted in order to create the cipher. Below, Table 4.3. benchmarks the total time it takes to encrypt each message using the Caesar Cipher and Table 4.4. benchmarks the total time it takes to decrypt each message using the Caesar Cipher.

Table 4.3. Caesar Cipher Average Encryption Times

Description: This table presents the benchmarks collected from the average milliseconds it takes to encrypt three different messages of different sizes, using three different keys of different sizes. The average is calculated by subtracting the beginning time from the end time and dividing it by 1000 (total number of times the loop iterates).

<table>
<thead>
<tr>
<th>Average time it takes to encrypt (ms)</th>
<th>Plaintext M1 (9 characters)</th>
<th>Plaintext M2 (25 characters)</th>
<th>Plaintext M3 (50 characters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key1 (of 2 digits)</td>
<td>0.009</td>
<td>0.020</td>
<td>0.021</td>
</tr>
<tr>
<td>Key2 (of 3 digits)</td>
<td>0.022</td>
<td>0.021</td>
<td>0.022</td>
</tr>
<tr>
<td>Key3 (of 4 digits)</td>
<td>0.022</td>
<td>0.022</td>
<td>0.022</td>
</tr>
</tbody>
</table>
Table 4.4. Caesar Cipher Average Decryption Times

**Description:** This table presents the benchmarks collected from the average milliseconds it takes to decrypt the three different messages encrypted previously (as illustrated in Table 4.3.) using the same three keys of three different sizes. The average is calculated by subtracting the beginning time from the end time and dividing it by 1000 since the loop iterates that number of times.

<table>
<thead>
<tr>
<th>Average time it takes to decrypt (ms)</th>
<th>Cipher M1</th>
<th>Cipher M2</th>
<th>Cipher M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key1 (of 2 digits)</td>
<td>0.009</td>
<td>0.016</td>
<td>0.021</td>
</tr>
<tr>
<td>Key2 (of 3 digits)</td>
<td>0.021</td>
<td>0.021</td>
<td>0.022</td>
</tr>
<tr>
<td>Key3 (of 4 digits)</td>
<td>0.021</td>
<td>0.021</td>
<td>0.022</td>
</tr>
</tbody>
</table>

The results from Test 1.1 shows the encryption average time did not vary from the average decryption time. For example, using Key2 to encrypt M1 takes 0.022 milliseconds and 0.021 milliseconds to decrypt the same message. Similar benchmarks can be seen for the rest of the results except for M1 and Key1, where it takes 0.009 milliseconds to encrypt and decrypt the message. The size of the message and size of key did not have an effect in the average time it takes to encrypt and decrypt a message. Therefore, using a larger key is recommended to encrypt and decrypt the data given that the average time will not increase.

**Test 1.2. Encrypting and Decrypting Messages with AES**

We created two different master key variants to test AES. Key1 is of size 128 bits (24 characters in Base64) and Key2 is of size 256 (44 characters in Base64). Similarly, to Test 1.1., Test 1.2. benchmarks the different average times it takes to encrypt and decrypt each message given one of two different size keys. Below, Table 4.5 benchmarks the time it takes to encrypt three different sized messages using the two keys for the AES
algorithm, and Table 4.6 benchmarks the time it takes to decrypt the previously encrypted messages implementing the same keys used to encrypt each message.

Table 4.5. AES Average Encryption Times

Description: This table displays the average times it takes the AES algorithm to encrypt all three plaintext messages individually using the two different size keys.

<table>
<thead>
<tr>
<th>Average encryption time (ms)</th>
<th>Plaintext M1 (9 characters)</th>
<th>Plaintext M2 (25 characters)</th>
<th>Plaintext M3 (50 characters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key1 (128 bits)</td>
<td>1.13</td>
<td>0.007</td>
<td>0.023</td>
</tr>
<tr>
<td>Key2 (256 bits)</td>
<td>1.521</td>
<td>0.007</td>
<td>0.022</td>
</tr>
</tbody>
</table>

Table 4.6. AES Average Decryption Times

Description: This table displays the average times it takes the AES algorithm to decrypt all three cipher messages individually using the two different size keys.

<table>
<thead>
<tr>
<th>Average decryption time (ms)</th>
<th>Cipher M1</th>
<th>Cipher M2</th>
<th>Cipher M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key1 (128 bits)</td>
<td>1.208</td>
<td>0.004</td>
<td>0.023</td>
</tr>
<tr>
<td>Key2 (256 bits)</td>
<td>1.663</td>
<td>0.004</td>
<td>0.022</td>
</tr>
</tbody>
</table>

Test 1.2., demonstrated that shorter messages will take a longer time to be encrypted/decrypted. As previously mentioned, the AES algorithm encrypts and decrypts data using 128-bit blocks. One hundred and twenty-eight bits is equivalent to 24 characters, therefore, plaintext and ciphers similar in size so the 128-bit blocks will take less to encrypt and/or decrypt. For example, encrypting and decrypting M2 (of 25 characters) takes 0.007 milliseconds to encrypt and 0.004 milliseconds to decrypt (no matter the size of the key). Plaintext and cipher of 9 characters takes longer to encrypt and decrypt because the AES algorithm needs to add padding to the end of the text to fulfill the 128-bit block criteria. For instance, plaintext of M1 takes 1.13 milliseconds to encrypt using the 128-bit key and 1.208 milliseconds to decrypt. The third message is of
50 characters structure, the AES algorithm divides the message into two blocks to encrypt and decrypt. Consequently, this exploits why it takes longer to encrypt and decrypt M3 in comparison to M2 and less than the time it takes for M1.

**Test 1.3. Authenticating Messages with ED25519**

This test measures the average time it takes for a message to be signed and verified using the elliptical curve ED25519. The test is running on a single Arduino Uno board where the program is uploaded. Again, three messages are used for this test and a private and public key are created to be used for the signature and verification of each message. Test 1.3. iterates 100 times, where the time begins immediately after the message has been signed and ends when the message has been verified.

Table 4.7 ED25519 Average Verification Time

**Description:** This table shows the average time it takes the ED25519 algorithm to verify each message of different size.

<table>
<thead>
<tr>
<th>Message</th>
<th>Average Verification Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 (9 characters)</td>
<td>9392.17</td>
</tr>
<tr>
<td>M2 (25 characters)</td>
<td>9413.34</td>
</tr>
<tr>
<td>M3 (50 characters)</td>
<td>9286.28</td>
</tr>
</tbody>
</table>

**Test 2: Average time it takes to send a simple message over Wireless Protocols**

Test 2 measures the average time it takes each environment (Wi-Fi and Bluetooth) to send a message from the client device to the server device and getting a response back from the server to the client. Each wireless module uses different size messages, please refer to the description below each environment for specific sizes and procedures.
Test 2.1. Round Trip Benchmark for Wi-Fi Environment

For the Wi-Fi environment, two sketches are created, one for the server and the other for the client. The client sends a message of 17 characters within a loop that iterates 1000 times. Once the server has received the message it responds with a six-character response. Table 4.7. displays the average time it takes the client to send and receive a response from the server using the \textit{millis()} method (an Arduino IDE function that returns the milliseconds passed since current program started running).

Table 4.8. Client/Server Average Communication Time (Wi-Fi)

\textbf{Description:} This table displays the average time it takes the client to send the same message to the server one thousand times. The time is calculated by subtracting the beginning time (set on the client side before the message is encrypted) from the end time (set on the client side after the client has decrypted the message the server sent back) and dividing it by 1000 (total number of iterations in the loop).

<table>
<thead>
<tr>
<th>Form of Communication</th>
<th>Average time to deliver message (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client- Server over Wi-Fi protocol</td>
<td>0.121</td>
</tr>
</tbody>
</table>

Test 2.2. Round trip benchmark for Bluetooth Environment

Due to time constraints, this section of the test is not expanded to an environment that uses two HC-05 modules, each communicating to an Arduino One acting as client and the other as server. However, one sketch is created to transmit the communication between the HC-05 and a Bluetooth Terminal on a smart phone. The sketch sent a 16-character message within a loop that iterates 1000 times. The sketch prints the beginning and end times of the loop using the \textit{millis()} method used for all the previous tests. The sketch sends the message and the terminal displays the message received.
Table 4. Client/ Server Average Communication Time (Bluetooth)

**Description:** This table shows the average time it takes the client and server to send and receive a message over the Bluetooth network. We used a Bluetooth terminal for Android phones (acted as server) and had an HC-05 module connected to an Arduino Board acting as client. The client measures the average time it takes to send a message to the Bluetooth terminal and for the Bluetooth Terminal to respond back to the client.

<table>
<thead>
<tr>
<th>Form of Communication</th>
<th>Average time to deliver message (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino to Bluetooth Terminal</td>
<td>0.003</td>
</tr>
</tbody>
</table>

Test 3: **Round trip benchmark for wireless environments using Caesar Cipher and AES**

Test 3 uses the Caesar Cipher and AES cryptographic algorithms to send data over Bluetooth and Wi-Fi environment we created in Tests 1 and 2. This test is broken into two sections, each section collects data from a different sensor (a microphone sensor and a photoresistor). Section I collects data from a microphone sensor LM393 (as illustrated in Figure 4.10. (a)) and Section II collects data from a photoresistor (as illustrated in Figure 4.10. (b)). Each section implements the two wireless environments (see Figures 4.11 and 4.12 for hardware connections) and each environment encrypts and decrypts data using both cryptographic protocols: Caesar Cipher and AES. For data sharing between client and server, the client sends the encrypted request and the client responds with the encrypted data. The Wi-Fi network is created through hotspot from a Samsung Galaxy S9 phone, functioning as a router rather than internet provider. This way, both Arduino boards connected to one another by saving the phone’s IP address and password.
Figure 4.10. Components of Microphone Sensor LM393 (a), Photoresistor (b) (Photo by: Ravi and Fotoresistor Sensor)

Figure 4.11. Hardware Components for Test 3, Section I of the Wi-Fi environment

Figure 4.12. Hardware Components for Test 3, Section II of the Wi-Fi environment
Test 3.1. Wi-Fi Environment with Caesar Cipher vs AES and Microphone Sensor

For this test, the Caesar Cipher is used to encrypt the communication between the client and server. The client sends an initial request of 20 characters that is encrypted using a secret key of value 13. The server receives the encrypted request, decrypts it using the key with value 13, and responded with a message of size greater than 26 characters (depending on the data collected by the microphone sensor). The average is calculated using the \texttt{millis()} method to measure the beginning and end of a loop that iterates 20 times. In other words, a single communication session is repeated 20 times. For the purpose of this thesis, a communication session begins when the client sends the request and it ends when the server has sent the encrypted responds back and client has decrypted the message as illustrated in Table 4.9.

For this test, AES is also used to encrypt the client/server communication. A 128-bit key is used to encrypt and decrypt the messages between the client and server. Again, a 20-character message is encrypted and sent by the client and the server responds with the encrypted data collected by the microphone sensor. The average time calculated using a \texttt{millis()} method to measure the beginning and end of a loop that iterates 20 times as illustrated in Table 4.9.
Table 4.10. Round trip benchmark in Wi-Fi client/server Environment (Microphone Sensor)

Description: This table shows the average time it takes the Caesar Cipher and AES to encrypt and encrypt the communication between client and server. Client sends the encrypted request and server decrypts the request, collects data from the microphone sensor, encrypts the data and sends it back for the client to decrypt.

<table>
<thead>
<tr>
<th>Wi-Fi Environment with Microphone Sensor</th>
<th>Average time it takes for Client-Server round trip (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caesar Cipher Algorithm</td>
<td>4,775.10</td>
</tr>
<tr>
<td>AES</td>
<td>5,002.05</td>
</tr>
</tbody>
</table>

Test 3.2. Wi-Fi Environment with Caesar Cipher vs AES and Photoresistor

For this test, AES is used to encrypt the client/server communication. A 128-bit key is used to encrypt and decrypt the messages between the client and server. Like in the Caesar Cipher section, a 20-character message is encrypted and sent by the client and the server responded with the encrypted data collected by the photoresistor. The average is calculated using a millis() method to measure the beginning and end of a loop that iterates 20 times. Section II also used Caesar Cipher to encrypt the communication between the client and server. Section II differs from Section I from the type of sensor used: here, a photoresistor is used to detect the levels of light within a room. The client sends an initial request of 20 characters that are encrypted using a secret key of value 13. The server received the encrypted request, decrypted it using the key with value 13 and responded with a message of size greater than 26 characters (depending on the data collected by the microphone sensor. The average is calculated using a millis() method to measure the beginning and end of a loop that iterates 20 times.
Table 4.1. Round trip benchmark in Wi-Fi client/server Environment (Photoresistor)

**Description:** This table shows the average time it takes the Caesar Cipher and AES to encrypt and encrypt the communication between client and server. Client sends the encrypted request and server decrypts the request, collects data from the photoresistor, encrypts the data and sends it back for the client to decrypt.

<table>
<thead>
<tr>
<th>Wi-Fi Environment with Photoresistor</th>
<th>Average time it takes for Client-Server round trip (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caesar Cipher</td>
<td>4,765.55</td>
</tr>
<tr>
<td>AES</td>
<td>5,002.05</td>
</tr>
</tbody>
</table>
CHAPTER 5: CONCLUSION

The thesis’s purpose is to collect benchmarks that could help future research and improvements of devices in the Internet of Things. With the benchmarks collected, we would like to share a review of the security abilities of these IoT devices in limited wireless environments. Securing the communication of devices within the IoT is crucial given that the data being transferred across different networks can be confidential. This thesis is tested in environments created with Arduino Uno boards (one of the many examples of devices in the IoT), two wireless protocols (Bluetooth and Wi-Fi), and sensors that captured data from their environment (photoresistor and microphone).

Our research demonstrated the difficulties of working with devices of low capabilities while simultaneous showing the potential of testing security measures in the Arduino Uno boards. The Arduinos have low memory and power capabilities, therefore, working with the devices proved to be challenging as the cryptographic algorithms had to be tailored specifically for the boards. However, it also allowed us to ensure that all communication had the capacity of being encrypted and decrypted, bringing a level of security that not many IoT devices execute.

Test 1.1 shows that having different key sizes for each cryptographic algorithm did not affect the average time it takes the algorithms to encrypt and decrypt the data. The Caesar Cipher takes very similar time, in milliseconds, to encrypt and decrypt the same message. For example, encrypting M2 with Key3 takes 0.022 milliseconds and 0.021 milliseconds to decrypt. The only variant that the results derived is in the encryption and decryption of M1 (9 characters) and Key1, while the other messages are encrypted and decrypted between 0.020 and 0.022 milliseconds, M1 is encrypted and decrypted in 0.009
milliseconds. The speed at which the M1 is encrypted and decrypted can be due to the size, given that M1 is the smallest message in size, and it also uses the smallest key, meaning it increased the speed at which it is encrypted and decrypted.

Test 1.2., on the other hand, demonstrated that a shorter message takes longer to encrypt and decrypt using AES algorithm. Given the nature of the algorithm, encrypting and decrypting data one block (128-bits in size) at a time, a shorter message requires more time since the algorithm adds padding to create a full block to then be able to encrypt/decrypt the data. For example, the 9-character message (M1) takes more than one millisecond to encrypt and decrypt no matter the size of the key used, while M2 takes less than 0.01 seconds. Based on the benchmarks collected in Test 1, we suggest using AES with a key of either size (128 or 256 bits) to encrypt data that can easily be divided into 128-bit blocks (data of size 25, 50, 100, etc), and using the Caesar Cipher with the largest size key to encrypt data as the size of the key had no effect in the time it takes to encrypt and decrypt.

Test 2 tested the average time at which each wireless module transmitted data from the client to the server. In the case of the Wi-Fi environment, a 17-character message is sent by the client and a 6-character response is received by the client from the server, takes an average time window of 0.121 milliseconds. The Bluetooth environment, on the other hand, sent and received a 16-character message from an Arduino Uno to a Bluetooth terminal in an Android phone taking only 0.003 milliseconds for the session to be complete.

Time constraints only allowed an implementation of Test 3 in a Wi-Fi environment. The Wi-Fi environment had two Arduino Unos, each with an ESP8266
module and sensors attached to the server side. Test 3 shows that there is a decimal
difference in the average time it takes each cryptographic algorithm to send data from
different sensors. For example, the Caesar Cipher takes 9.55 milliseconds longer to
encrypt and decrypt data when it is being collected from the microphone sensor and
compared to the photoresistor. The AES algorithm on the other hand, takes the same time
to encrypt and decrypt the data from both sensors. In comparison, the AES algorithm
takes more than 200 milliseconds longer to encrypt/decrypt data than the Caesar Cipher.
Based on these results, we suggest using the Caesar Cipher if there is a preference for
speed, while if greater security measures are preferred over speed, AES would be
recommended.

For future implementations we suggest expanding Test 3 to a Bluetooth
environment where a master HC-05 module acts as the client and the slave HC-05
module as the server. In addition, performing Test 3 with the elliptical curve Ed25519 to
signature and verification aspects to the test. Implementing small degrees of security
within these devices that makeup the IoT can add greater privacy to the data being stored
and transferred among these environments.
REFERENCES


