

**Development of a Secure Raspberry Pi Based Home Surveillance System**

A dissertation submitted in partial fulfilment of the requirements of Glasgow Caledonian University for the degree of Master of Science in Big Data Technologies

Christopher Dillon

S1514278

**“Except where explicitly stated, all work in this report, including the appendices, is my own original work and has not been submitted elsewhere in fulfilment of the requirement of this or any other award”**

**Signed: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_**

# Abstract

With the recent rise of the Internet of Things, smart devices are becoming ubiquitous and synonymous with modern living. Increasing dependence on these devices along with frequent media scares have increased both the media and publics concerns about the security of these devices.

Smart home surveillance is a growing market within IoT which holds greater concern as many of these devices are within the confinements of an individual’s private home. With most home surveillance systems available on the market hosting their data on the cloud, this further increases the scope of potential security vulnerabilities and gives individuals with malicious intent an additional target.

This thesis proposes an alternative local network and storage based surveillance system using a Raspberry Pi 3b Model to combat the potential security vulnerabilities within mainstream systems. Multiple face recognition algorithms are evaluated and factors such as memory consumption, CPU usage and accuracy are considered.

An additional mobile application is developed, to interact with the Raspberry Pi using the MQTT communication protocol. This includes the receiving of images which are then stored within an SQLite Database. Using the hybrid application framework Xamarin, the application was successfully developed for both iOS and Android.

With security in mind, state of the art security strategies was implemented to secure the application, Raspberry Pi and network communication between the devices. Strategies include *salt* password hashing, database encryption, certificate-based MQTT communication and firewall incorporation.

Finally, the system was tested against various local network attacks, including Direct Denial of Service and Man in The Middle attacks to evaluate the defences in the place to prevent such attacks.

# Acknowledgments

I would like to thank my Supervisor, Dr Anna Lito Michaela, for her continuous support throughout the project. Her guidance and suggestions were paramount in the success in this project along with the continual feedback.

Additionally, I would like to thank both my family and friends for the support and understanding of my lack of availability during the writing of my thesis. Finally, I would like to thank all my colleagues at my past job. The continuous support and encouragement was a key variable in the completion of my degree.

Table of Contents

[1.0 Introduction 12](#_Toc17899654)

[1.1 Background 12](#_Toc17899655)

[1.2 Problem Description 14](#_Toc17899656)

[1.3 Project Objectives 14](#_Toc17899657)

[2.0 Literature Review 16](#_Toc17899658)

[2.1 Internet of Things Architecture 16](#_Toc17899659)

[2.1.1 Perception Layer 16](#_Toc17899660)

[2.1.2 Network Layer 17](#_Toc17899661)

[2.1.3 Application Layer 17](#_Toc17899662)

[2.2 Network Communication Protocols 17](#_Toc17899663)

[2.2.1 Bluetooth 18](#_Toc17899664)

[2.2.2 Wi-Fi 18](#_Toc17899665)

[2.2.3 ZigBee 19](#_Toc17899666)

[2.2.4 HART 19](#_Toc17899667)

[2.2.5 Comparative Analysis 19](#_Toc17899668)

[2.3 Application Communication Protocols 19](#_Toc17899669)

[2.3.1 Message Queue Telemetry Transport 20](#_Toc17899670)

[2.3.2 Advanced Message Queueing Protocol 21](#_Toc17899671)

[2.3.3 Constrained Application Protocol 22](#_Toc17899672)

[2.3.4 REdis Serialization Protocol 22](#_Toc17899673)

[2.3.5 Comparative Analysis 23](#_Toc17899674)

[2.4 Data Exchange Formats 23](#_Toc17899675)

[2.4.1 XML 24](#_Toc17899676)

[2.4.2 JSON 24](#_Toc17899677)

[2.4.3 Protocol Buffers 25](#_Toc17899678)

[2.4.4 Comparative Analysis 25](#_Toc17899679)

[2.5 Security Threats 26](#_Toc17899680)

[2.5.1 Denial Of Service Attack 26](#_Toc17899681)

[2.5.2 Man in The Middle Attack 27](#_Toc17899682)

[2.5.3 Brute Force Attacks 28](#_Toc17899683)

[2.5.4 Physical Tampering 28](#_Toc17899684)

[2.5.5 Malicious Code Attack 29](#_Toc17899685)

[2.6 Cryptography Options 29](#_Toc17899686)

[2.6.1 Symmetric Encryption 29](#_Toc17899687)

[2.6.2 Asymmetric Encryption 29](#_Toc17899688)

[2.6.3 Hash Encryption 30](#_Toc17899689)

[2.6.4 Salt Encryption 31](#_Toc17899690)

[2.7 Message Integrity 31](#_Toc17899691)

[2.7.1 Hash Algorithms 31](#_Toc17899692)

[2.7.2 Message Authentication Code Algorithms 31](#_Toc17899693)

[2.7.3 Digital Signatures 32](#_Toc17899694)

[2.7.4 Comparative Analysis 32](#_Toc17899695)

[2.8 Facial Recognition 32](#_Toc17899696)

[2.8.1 Haar Cascade 32](#_Toc17899697)

[2.8.2 Histogram of Oriented Gradients 33](#_Toc17899698)

[2.9 Security Guidelines for Internet of Things 35](#_Toc17899699)

[3.0 Technology Review 36](#_Toc17899700)

[3.1 Hardware Selection 36](#_Toc17899701)

[3.2 Operating System Selection 36](#_Toc17899702)

[3.2.1 Raspberry Pi Operating System 36](#_Toc17899703)

[3.2.2 Cyber Security Operating System 36](#_Toc17899704)

[3.3 Programming Language Selection 36](#_Toc17899705)

[3.4 Library Selection 37](#_Toc17899706)

[3.4.1 Python Library Selection 37](#_Toc17899707)

[3.4.2 C# Library Selection 38](#_Toc17899708)

[3.5 Supporting Software Review 39](#_Toc17899709)

[4.0 System Requirements 40](#_Toc17899710)

[4.1 Hardware and Software Requirements 40](#_Toc17899711)

[4.2 Functional & Non Functional Requirements 41](#_Toc17899712)

[5.0 Methodology 42](#_Toc17899713)

[5.1 Development Approach 42](#_Toc17899714)

[5.2 Main Flow Design & Overview 43](#_Toc17899715)

[5.3 Surveillance System Development 44](#_Toc17899716)

[5.3.1 Gathering & Encoding Face Dataset 44](#_Toc17899717)

[5.3.2 Real-Time Face Recognition 46](#_Toc17899718)

[5.3.3 Power Outage Considerations 47](#_Toc17899719)

[5.3.4 Security Implementation 48](#_Toc17899720)

[5.4 Application Development 49](#_Toc17899721)

[5.4.1 Initial Class Diagram Model 49](#_Toc17899722)

[5.4.2 Account Creation and Login 49](#_Toc17899723)

[5.4.3 Database Development 52](#_Toc17899724)

[5.4.4 Handling Received Images 53](#_Toc17899725)

[5.4.5 MQTT Configuration Settings 53](#_Toc17899726)

[5.5 MQTT Communication 54](#_Toc17899727)

[5.5.1 Surveillance System Communication 55](#_Toc17899728)

[5.5.2 Application MQTT Communication 55](#_Toc17899729)

[5.5.3 MQTT Communication Security 57](#_Toc17899730)

[6.0 Experiment Setup 60](#_Toc17899731)

[6.1 System Testbed 60](#_Toc17899732)

[6.1.1 Latency 60](#_Toc17899733)

[6.1.2 System Accuracy & Performance Setup 61](#_Toc17899734)

[6.3 Cyber Security Testbed Setup 62](#_Toc17899735)

[6.3.1 Dictionary Attacks 62](#_Toc17899736)

[7.0 Evaluation 63](#_Toc17899737)

[7.1 Surveillance System Performance 63](#_Toc17899738)

[7.1.1 Accuracy of Algorithms 63](#_Toc17899739)

[7.1.2 Computational Stress 64](#_Toc17899740)

[7.1.3 Memory Consumption 65](#_Toc17899741)

[7.1.4 System Latency 66](#_Toc17899742)

[7.3 Portability 66](#_Toc17899743)

[7.4 Security 67](#_Toc17899744)

[7.4.1 Brute Force 67](#_Toc17899745)

[7.4.2 Denial Of Service 67](#_Toc17899746)

[7.4.3 Man In The Middle 67](#_Toc17899747)

[7.4.4 IoT Security Foundation Guidelines 68](#_Toc17899748)

[8.0 Conclusion 69](#_Toc17899749)

[8.1 Project Limitations 69](#_Toc17899750)

[8.2 Future Work 69](#_Toc17899751)

[8.3 Closing Statement 69](#_Toc17899752)

[9.0 References 70](#_Toc17899753)

[10.0 Appendices 76](#_Toc17899754)

[Appendix A – Raspberry Pi Specifications 76](#_Toc17899755)

# Table of Figures

[Figure 1: Smart Home Security Revenue forecast worldwide (Blumtritt, 2019) 12](#_Toc17899756)

[Figure 2: Raspberry Pi Model 3b 13](#_Toc17899757)

[Figure 3: IoT three layered architecture in relation to a home surveillance system 17](#_Toc17899758)

[Figure 4: MQTT Architecture using a Raspberry Pi as both a Publisher and Broker 21](#_Toc17899759)

[Figure 5: Basic AMQP protocol process with exchange and queue system 21](#_Toc17899760)

[Figure 6: Basic GET/Response between a CoAP Client & Server 22](#_Toc17899761)

[Figure 7: Code Snippet showing basic XML Schema 24](#_Toc17899762)

[Figure 8: Code snippet showing basic JSON Schema 24](#_Toc17899763)

[Figure 9: A DDoS attack being carried out using infected PCs (slaves) 26](#_Toc17899764)

[Figure 10: MITM attack showing an intercepted connection from a malicious attacker 27](#_Toc17899765)

[Figure 11: Computer running a malicious script to attempt logins on a website 28](#_Toc17899766)

[Figure 12: Example of plain text being encrypted using symmetric key encryption 29](#_Toc17899767)

[Figure 13: Example of plain text being encrypted using Asymmetric key encryption 30](#_Toc17899768)

[Figure 14: SHA-1 & MD5 brute force successful attack times (Putri Ratna et al., 2013) 30](#_Toc17899769)

[Figure 15: Most commonly detect features in the Haar Cascade algorithm 33](#_Toc17899770)

[Figure 16: Incremental Software Development Lifecycle for a Home Surveillance System 42](#_Toc17899771)

[Figure 17: Main Interaction of the Surveillance System and Smart Phone Application 43](#_Toc17899772)

[Figure 18: Example of using face\_recognition support library to receive faces 45](#_Toc17899773)

[Figure 19: UML Class Diagram highlighting the relationships between the user and MQTT interaction 49](#_Toc17899774)

[Figure 20: Android Emulator showing fingerprint verification 52](#_Toc17899775)

[Figure 21: Entity Relationship showing overall database structure 53](#_Toc17899776)

[Figure 22: Android Emulator showing MQTT configuration settings page 54](#_Toc17899777)

[Figure 23: Mosquitto broker (RPi) interacting with Paho & M2Mqtt clients 55](#_Toc17899778)

[Figure 24: App showing successful connection on three different security layers 56](#_Toc17899779)

[Figure 25: MQTT Communication secured using Certification and TLS 59](#_Toc17899780)

[Figure 26: Kali Linux Kernel showing Dictionary Attack using 'Hydra' Software 62](#_Toc17899781)

[Figure 28: Difference in Computational Stress between the HOG and Haar algorithms 65](#_Toc17899782)

[Figure 29: Difference in Memory Consumption between the HOG and Haar algorithms using 10 encoded images 65](#_Toc17899783)

# Table of Tables

[Table 1: Comparison of common IoT Network Communication Protocols 19](#_Toc17899784)

[Table 2: Features of most common IoT Application Communication Protocols 23](#_Toc17899785)

[Table 3: Serialized size in bytes (Sumaray and Makki, 2012) 25](#_Toc17899786)

[Table 4: Average serialization time in milliseconds (Sumaray and Makki, 2012) 25](#_Toc17899787)

[Table 5: Average deserialization time in milliseconds (Sumaray and Makki, 2012) 25](#_Toc17899788)

[Table 6: Relevant IoT best practices in relation to the project 35](#_Toc17899789)

[Table 7: Hardware Requirements for project 40](#_Toc17899790)

[Table 8: Software Requirements for project 40](#_Toc17899791)

[Table 9: Core Functional Requirements 41](#_Toc17899792)

[Table 10: Non-Functional Requirements 41](#_Toc17899793)

[Table 11: Comparison of hashing algorithms using the term 'password' 50](#_Toc17899794)

[Table 12: Performance matrix table giving four possible outcomes 63](#_Toc17899795)

[Table 13: HOG Algorithm Confusion Matrix using different volumes of encoded images 63](#_Toc17899796)

[Table 14: Haar-Cascades Algorithm Confusion Matrix using different volumes of encoded images 64](#_Toc17899797)

[Table 15: Accuracy of algorithms using different volumes of images encoded 64](#_Toc17899798)

[Table 16: Comparison of CPU & Memory usage between the HOG & Haar algorithms 66](#_Toc17899799)

[Table 17: Results of security measures to prevent against a Brute Force attack 67](#_Toc17899800)

[Table 18: Proposed solutions for IoT Security best practices 68](#_Toc17899801)

# Table of Code Fragments

[Code Fragment 1: Python code handling capturing images using ‘k’ key 44](#_Toc17899802)

[Code Fragment 2: Python code showing facial encodings for each individual carried out 45](#_Toc17899803)

[Code Fragment 3: Python code showing frame modification, speeding up real-time processing 46](#_Toc17899804)

[Code Fragment 4: Python code showing face comparison in real time 46](#_Toc17899805)

[Code Fragment 5: Python code showing image capture of unidentified subject 47](#_Toc17899806)

[Code Fragment 6: C# Pseudocode showing password encryption with SHA512 + Salt 51](#_Toc17899807)

[Code Fragment 7: C# Code showing login validation by comparing two hash values 51](#_Toc17899808)

[Code Fragment 8: C# Pseudocode on handling messages received from MQTT 53](#_Toc17899809)

[Code Fragment 9: Python code showing publish method for Paho Client 55](#_Toc17899810)

[Code Fragment 10: C# Code handling different levels of secure MQTT M2Mqtt client creation 57](#_Toc17899811)

[Code Fragment 11: Modified version of script shown in Code Fragment 6 60](#_Toc17899812)

[Code Fragment 12: Modified version of 'surveillance.py' script used to record accuracy 61](#_Toc17899813)

# Table of Abbreviations

|  |  |
| --- | --- |
| IoT | Internet of Things |
| HART | Highway Addressable Remote Transducer |
| MQTT | Message Queue Telemetry Transport |
| QoS | Quality of Service |
| AMQP | Advanced Message Queueing Protocol |
| CoAP | Constrained Application Protocol |
| RESP | REdis Serialization Protocol |
| XML | Extensible Markup Language |
| JSON | JavaScript Object Notation |
| DoS | Denial of Service |
| DDoS | Direct Denial of Service |
| MitM | Man in the Middle |
| AES | Advanced Encryption Standard |
| RSA | Rivest–Shamir–Adleman |
| MD5 | Message-Digest Algorithm |
| SHA | Secure-Hash Algorithm |
| HOG | Histogram of Oriented Gradients |
| CNN | Convolutional Neural Network |
| OS | Operating System |
| RPi | Raspberry Pi |
| CSV | Coma Separated Values |
| TP | True Positive |
| TN | True Negative |
| FP | False Positive |
| FN | False Negative |

# 1.0 Introduction

## 1.1 Background

With the recent explosion of the Internet of Things (IoT), there is a rising concern by the general public on the security of these devices. In 2015, HP published a report which highlighted that out of the ten home security systems analysed, 100% of them had security vulnerabilities such as not requiring strong passwords and only one offered two-factor authentication (HP, 2015). Concerning reports such as the aforementioned along with recent media scares such as baby monitors being hacked with the hacker being able to speak through the baby monitor and threatening the parents (Wang, 2018), have brought these issues into the public’s attention, raising concern about the validity and/or the security of these devices.

This is concerning enough that the UK Government poses to introduce new legislation based on Internet of Things (IoT) security, requiring products to pass security requirements before being sold and introducing mandatory labelling schemes for each product, classifying their security (Ashford, 2019). The industry of Smart Security expects a revenue forecast of 28 million by the end of 2023 (shown in Figure 1), further highlighting the importance of development of secure IoT devices.



Figure 1: Smart Home Security Revenue forecast worldwide (Blumtritt, 2019)

With mainstream IoT devices being consistently hosted on the cloud, this offers individuals with malicious intent another platform to attack IoT devices. Recently, Google Cloud, one of the cloud vendor giants was taken down in a possible cyber-attack (Merriman, 2019), highlighting the potential unreliability of Cloud-hosted services. Organisations that depended on Googles Cloud, including security company Nest, who had their services taken down for over 4 hours. This is problematic in an area such as face-recognition based surveillance; if data cannot be accessed from the Cloud, the whole system becomes non-functional.

As an alternative to the current mainstream method of hosting on the cloud, a Raspberry Pi-based security system utilising local storage is proposed in this thesis. The Raspberry Pi (RPi), being a small credit card-sized computer, has significant capabilities in home smart technology, only limited to its processing speed and memory. The model board can be shown in Figure 2. The proposed system involves interaction between various smart phone devices running on both iOS and Android with the RPi via the MQTT communication protocol. This would involve the RPi using a face recognition algorithm to detect if an intruder or unidentified person is at the door of the home and capturing an image. Training using images of family or friends can allow for them to be individually identified and therefore not be labelled as unknown to the system. Images captured would then be sent to the app and then be viewable to those who have access to the application.



Figure 2: Raspberry Pi Model 3b

Building a custom made home surveillance system poses various security challenges similar to all IoT devices, which occur at all layers of the IoT architecture. The most commonly agreed upon being the Perception, Network and Application layer (Burhan et al., 2018). Most of the strategies incorporated into this thesis persist around the Application and Network layers of this architecture. State of the art strategies are proposed such as Certificate-based authentication and implementing password storage strategies such as *salt* password hashing. Protection against common IoT attacks such as Denial of Service (DoS) and Man in The Middle attacks are investigated to propose a solution to these security threats. By investigating the best security practices, a model can be proposed to build future similar systems.

## 1.2 Problem Description

Modern IoT-based security systems pose various security risks. With the majority of these services using cloud technology, there is a larger scope of security concerns covering the full system stack. The development of a home security system using an RPi model is proposed to create a system with a smaller scope of security vulnerabilities using local data storage to reduce information exposure over the system. The RPi will interact with a smartphone app, with each family member having access to the application. This means security can be focused down to the perception, network and application layers of the IoT architecture.

This project aims to tackle the following areas of security:

* Data Integrity. It is important to know when transferring of files that the data has not been tampered with by a third party. Tampering of files could lead to various types of malware or viruses that did not originate from the original sender.
* Message Authentication. Verifying that the individual or party who is sending data is who they say they are. This covers both the recipient and sender of the message as sensitive data must be secured in both instances.
* Storage security. There are various strategies in password storage to prevent unauthorised access to passwords such as implementation of various forms of one-way cryptography i.e. hashing. Countermeasures to this strategy such as dictionary attacks must be investigated to learn how to protect against such cyber threats.
* Disk storage security. Unauthorised access to the physical device is definitely a possibility. It is important to secure the contents of the disk in an unlikely event of physical tampering of the device.

These requirements will be fulfilled in accordance with the best practices outlined by the IoT Security Foundation (IoT Security Compliance Framework, 2018). The organisation details the golden standards in IoT security which can be referenced to ensure the device is secured to the highest standard.

## 1.3 Project Objectives

This project aims to achieve the following objectives:

1. Identify from the available literature, state of the art guidelines for an IoT-based home surveillance system. With multiple network and communication protocols being available for IoT devices, it is important to determine the most secure and standard for this type of home system.
2. Review the IoT architecture to understand the security and design flaws for each layer. The perception, network, and application layer currently pose various security challenges for IoT and must be addressed individually. This would involve reviewing papers of similar systems and determining the possible security flaws to give a greater understanding of the possible security vulnerabilities to the proposed system.
3. Determine the most appropriate data exchange format. With there being multiple data formats available for application communication between devices, the most suitable is selected, being determined from variables such as serialization time, deserialization time and message size.
4. Ensure the security standards are in alignment with the guidelines set by IoT security foundation (IoT Security Compliance Framework, 2018). By comparing the system to the best practices standard, it can be ensured that the system meets security requirements. It is frequently debated that security is always traded for convenience (Cherecwich, 2019).. Factors such as latency should also be considered with the implementation of security.
5. Identify common security threats to IoT-based devices. It is important to understand the most prominent cyber-attacks carried out on these devices to prepare a defence against them. This would include carrying out various cyber-attacks on the device and measuring how they protect against such attacks.
6. With devices such as the Pi being limited by its computational power, it is important to determine the most appropriate face recognition methodology suitable for a small device. There are various types of algorithms available to choose from. Factors such as CPU stress and memory consumption are a major consideration when it comes to selecting an algorithm.
7. Development of the surveillance system. The system must be able to recognise faces of individuals and appropriately label them while being able to detect unidentified faces which can then be captured and sent to the smart phone devices. Accuracy and latency are two of the major factors as the system must detect in real time.
8. Development of the smart phone application. The application must be able to interact with the Pi and receive images in a timely manner. As this data is sensitive, security measures must be in place to verify the authenticity of the individual using the device.
9. Evaluate functionality. The system must be fully functional upon evaluation as a security camera within a home is a sensitive area. This would involve software testing such as unit tests to evaluate and draw conclusions on each part of the system.

# 2.0 Literature Review

Raspberry Pi-based home surveillance systems are not a new concept but have posed various challenges relating to the security of the system. In a 2018 paper (Pawar and Umale, 2018), a home surveillance system was created using the RPi utilising a PIR sensor to detect movement and sending an email once detected. The disadvantage of this system is that there is no way to detect who is at the door and no categorisation of whether the individual is a threat or not. Animals or other movements could also trigger the sensor leading to a false alert. The system also uses email as a form of alert which is problematic as email can be accessed anywhere and offers little or no form of authenticating the identify of an individual. A similar system was designed using face recognition technology or a password to enter a home (Hussein and Al Mansoori, 2017). Again an email is sent when there is an unknown individual at the door, posing the same problems as the previously mentioned paper. The proposed system hopes to address these challenges by using secure transport methods, two-factor authentication and limiting the number of potential security vulnerabilities to a single smartphone application.

This section covers the overall IoT architecture and the current security issues present in each layer. State of the art network and communication protocols are reviewed as well as security protocols to determine the most suitable protocols for the proposed system. Various security threats are reviewed in order to understand the potential attacks that can be carried out on the system. Cybersecurity methods such as cryptography and salt hashing are reviewed to determine their suitability for the project. Finally, facial recognition algorithms are reviewed with the computation limitations of the Pi in mind.

## 2.1 Internet of Things Architecture

IoT security must address three main levels of architecture that can be attacked with malicious intent. As detailed in a 2018 paper, the most basic agreed-upon architecture consists of three layers: Perception Layer, Network Layer and Application Layer (Burhan et al., 2018). This can be seen in Figure 2. For Cloud-based systems, an additional Processing layer is also considered (Aziz and Haq, 2018).

### 2.1.1 Perception Layer

This layer consists of the physical sensors which collect information and identify objects. Commonly used sensors include RFID, barcode & cameras. In this experiment, the security camera is the main focus of this layer and can pose multiple potential security issues. As highlighted by Aziz and Haq, eavesdropping is definitely the most concerning security issue in this layer. Having unauthorised access to video footage is a serious breach of privacy. This is accomplished by taking advantage of insecure modes of transmission, granting access to a third party without authorisation.

### 2.1.2 Network Layer

The network layer is responsible for the transmission of data, acting as a bridge between the perception and application layer. This involves carrying and transmitting information through a wireless network which poses a set of security challenges. These include various cyber-attacks such as Man in the Middle (MITM) attacks and Denial of Service (DoS) as later detailed in Section 2.5. The layer acts as a Central Nervous System for the whole system. In this scenario, the network layer consists of a standard wireless home network. Mainstream protocols such as Wi-Fi and ZigBee are often used at this layer, as well as lightweight protocols such as MQTT.

### 2.1.3 Application Layer

This layer is used to define all applications that use the data generated at the perception layer. Common examples of this layer include smart homes, smart cities, smart health, etc. Security threats to this layer would include attempts of password theft using attacks such as Dictionary attacks or Brute Forcing. This layer utilises the data gained from the previous layers and allows the user of the application to enjoy its benefits. In this experiment, the application layer would consist of the smartphone app used to operate the door and receive images/information from the security camera. The user interface of the application is always a significant component of this layer.



Figure 3: IoT three layered architecture in relation to a home surveillance system

## 2.2 Network Communication Protocols

IoT devices differs from traditional computing devices in terms of network communication, utilising additional communication technologies such as ZigBee and Bluetooth, compared to traditional computers solely relying on Wi-Fi-based communication. The most commonly used protocols are reviewed in this section and critically analysed against these requirements to select the protocol to be used for in this thesis. Factors such as power consumption are not considered as security and functionality are valued over this variable.

The following design requirements were identified for the network communication protocol used in this thesis:

* An adequate range to reach the RPi carrying out the surveillance. Limited communication range defeats the purpose of this system as the idea is to access the surveillance system both remotely and locally.
* Communication protocol is supported by both the RPi and modern smart phones. The communication protocol must be accessible to both smart devices and the RPi as this is how the images will be sent.
* High levels of security capability. Having insecure communication between the devices opens up a lot of possible security vulnerabilities and could grant unauthorised individuals access to sensitive data.

* Capable of fast data transmission. As the camera will detect in real time it is important for the users of the app to receive the images immediately as there is little use in knowing who was at the door hours ago.

### 2.2.1 Bluetooth

Bluetooth is used to communicate between two devices within a short distance. It provides a layer of encryption by converting a message into ciphertext before sending it to the receiver device. This message, cannot be understood by other devices except those which have the rights to see the message. The sender must always get permission rights from the receiving device before the message can be sent. This is done through the sender device requesting permission to send data to the receiver device, once agreed to, the devices can then communicate within the short distance. Bluetooth has a max signal rate of 1mb/s and an average range of 10 meters (Lee, Su, and Shen, 2007). This communication protocol has some downfalls as is prone to packet sniffing. A recent 2019 paper highlights this with the development of a Bluetooth sniffer called BlueEar, boasting a 90% packet retention rate in real-world environments. Though it should be noted the authors offer countermeasures to this device, which can be effective in reducing the packet retention rate to as low as 20%. Although promising, this suggests Bluetooth is still not a secure method of network communication as 20% of packet data is still being stolen from the devices.

### 2.2.2 Wi-Fi

Wi-Fi is a wireless communication network that transmits communication in the form of radio signalling. This is the most common type of communication used in modern homes which is problematic as it contains multiple security vulnerabilities. The main issue being, by default, there is no encryption mechanism. This leaves the network prone to network cyberattacks such as Denial of Service attacks. It is well documented that Wi-Fi is one of the most commonly used network protocols in smart homes (Alam, Reaz and Ali, 2012). Wi-Fi has the added benefit of being the fastest of all common communication technologies with a recent report showing the average speed being 16.51Mbps (Cable, 2019). The average range of a Wi-Fi connection is 100 meters (Lee, Su, and Shen, 2007).

### 2.2.3 ZigBee

ZigBee is a communication protocol, which utilises small, low-power and cost radios. It is frequently utilised in the smart homes in conjunction with hubs such as the Amazon Echo (ZigBee Alliance, 2019). The technology is very limited in terms of signal rate with a rate of 250Kb/s (Lee, Su, and Shen, 2007), making it unsuitable for video transmission. The transmission range is rated between 10 and 100 meters. In the paper (Romashchenko, Brutscheck, and Chmielewski, 2018), a secure surveillance system was created based on ZigBee and AES encryption, showing success against brute force attacks.

### 2.2.4 HART

Highway Addressable Remote Transducer or Hart for short is a communication protocol used primarily in an industrial setting due to its ability to accurately encode and decode communication signals in settings with noisy or harsh environments (Treacy, 2019). It is considered the global standards for digital communication across analogue wires between smart devices and monitoring/control systems (Instrumentation and Control Engineering, 2018).

### 2.2.5 Comparative Analysis

As shown through the comparative analysis in Table 1, Wi-Fi meets all the identified design requirements and is the most commonly used form of network communication. Thus, it is most suitable for the system proposed in this thesis. Being commonplace in homes along with its high range and fast data transmission feed, gives Wi-Fi a significant advantage over its networking counterparts.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Requirement* | *Bluetooth* | *Wi-Fi* | *ZigBee* | *Hart* |
| High range |  |  |  |  |
| Commonplace in home |  |  |  |  |
| Security |  |  |  |  |
| Fast Data Transmission |  |  |  |  |

Table 1: Comparison of common IoT Network Communication Protocols

## 2.3 Application Communication Protocols

Application communication protocols are lightweight low power and low data transmission in comparison to their network communication counterparts. These forms of communication protocols were designed specifically for IoT devices (Karagiannis et al., 2015). This section reviews some of the most commonly used application communication protocols. This communication is used within the application layer of the IoT architecture to allow microservices to communicate between them the acquired and/or processed data for higher levels of processing or information inference.

### 2.3.1 Message Queue Telemetry Transport

MQTT is the most commonly used application layer communication protocol, being lightweight and using a publish-subscribe model. The protocol is mainly used where a small code footprint is required i.e. sensor data and where bandwidth is limited. With the maximum amount of data transferred being 256MB (Rastovich, 2015), speed is the priority with this protocol. It is based on the client/server architecture; the server is responsible for handling the client's request for receiving or sending data between each other. This means when a device sends data to the broker (client) it is ‘publishing’ data for any ‘subscribers’ listening to that particular topic.

It supports three levels of Quality of Service (QoS), with each having different volumes of packets exchanged. The QoS levels have the following features:

* QoS Level 0 sends a message once and does not check if the message arrived at its given destination. This is the most basic form of message transport and can be used when there
* QoS Level 1 improves upon this by sending the message at least once and checks the delivery status using PUBACK, a status check message. If the PUBACK is lost, this can result in the same message being sent twice, due to no confirmation of the original message.
* QoS Level 2 sends the message exactly once by utilising a 4-way handshake between the client and broker. This has the benefit of there being no possibility of the message being lost but can cause additional delays.

In a correlation analysis measuring lost and delay of MQTT QoS levels, it was found that for Levels 0 and 1, the average packet loss was around 0.90% to 1.40% whereas packet loss only occurred at maximum, 0.18% with QoS level 2 (Lee et al., 2013). Although QoS level 2 did have the highest end-to-end delay, this was at maximum one second, suggesting QoS Level 2 would be most suitable for the proposed system. Sharing the same attributes as MQTT, SMQTT introduces an encryption/decryption protocol to enhance security (Singh et al., 2015). Figure 3 highlights the communication flow between an example Pi Broker acting as both a client (being able to send messages) and the server (receiving MQTT messages). Figure 4 also shows multiple smart phone clients subscribed to a specific topic awaiting a response.



Figure 4: MQTT Architecture using a Raspberry Pi as both a Publisher and Broker

### 2.3.2 Advanced Message Queueing Protocol

AMQP is an open standard application layer protocol for middleware messaging. It has additional features such as message orientation, switching, reliability and queueing. Both request/response and publisher/subscriber models are available with this protocol. Communication via this protocol is done through either the publisher or consumer creating an “exchange” with a given name, that name can then be broadcasted for both publishers and consumers to discover each other (Figure 5). Along with this exchange, the consumer creates a “queue”, attaching it to the exchange. Messages that are received by the exchange are matched to the queue via a binding process. AMQP offers the most flexibility when it comes to message exchange, offering multiple methods such as message by topic (similar to MQTT), directly, based on headers or in a fan-out form. In terms of QoS, AMQP offers two types of delivery of messages: Unsettle Format and Settle Format, being similar to MQTT’s Level 0 and 1.

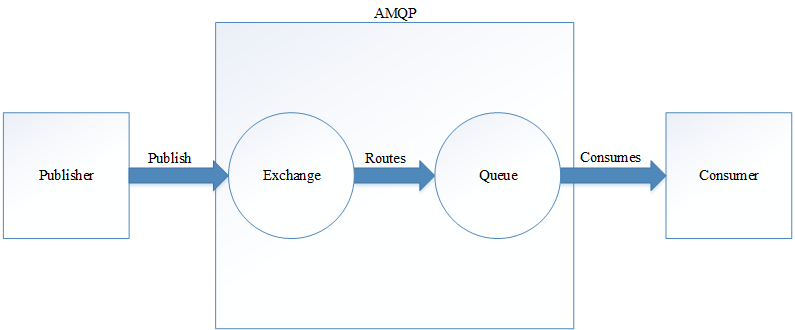


Figure 5: Basic AMQP protocol process with exchange and queue system

### 2.3.3 Constrained Application Protocol

CoAP is a stateless client/server application protocol based on the REST framework. This is based on a request/response model using the request types ‘GET’, ‘POST’, ‘PUT’ and ‘DELETE’, being similar to the standard HTTP protocol. CoAP uses a Universal Resource Identifier (URI) instead of topics seen in protocols such as MQTT. This is using a similar publish/subscribe method. The publisher sends new data to the URI, and all the subscribers are notified about the new value indicated by the URI, similar to MQTT. CoAP utilises UDP as a transport protocol and DTLS for security (Ludovici, Moreno and Calveras, 2013). Using UDP instead of TCP does create less reliability as there is no guaranteed delivery of a packet. To combat this; there is the option of “confirmable” and “non-confirmable” messages to provide a QoS architecture. Confirmable messages work by being acknowledged by the receiver via an ACK packet (Figure 6). Non-confirmable messages have no such system in place.

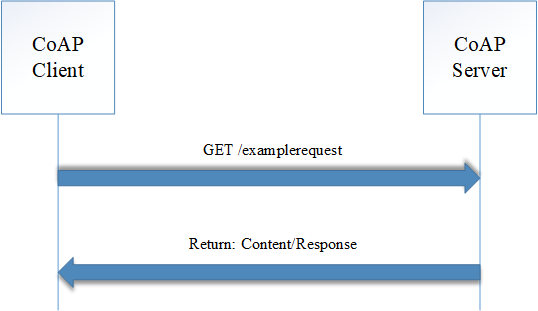


Figure 6: Basic GET/Response between a CoAP Client & Server

### 2.3.4 REdis Serialization Protocol

RESP is an application protocol designed specifically for the Redis key-value database (Redis.io, 2019). The main feature of this protocol is its ability to serialize data types including strings, integers, and arrays. Similar to MQTT and AMQP, it is primarily used with a TCP connection. A request/response model is also used and functions by sending commands to the server. This command is then received and once processed, a response is sent back to the client. RESP has a unique feature in which responses can be scheduled; a client can send multiple commands at once and wait for a response at a later time. Another area where this protocol differs is when a client subscribes to a public/subscribe channel, it becomes a push protocol, in which the client will automatically receive messages without having to send commands.

### 2.3.5 Comparative Analysis

As shown in Table 2, MQTT seems to offer the most in terms of reliability of message delivery. Utilising QoS Level 2 has the guarantee the message is not lost. The other protocols offer no such service, with CoAP and AMQP only offering the equivalent to Level 1 QoS. Transport protocol is also a consideration when selecting a communication protocol, as TCP is much more reliable than its UDP counterpart. It was also shown when it comes to sending small bytes of data similar to the payloads of most IoT devices, MQTT appears to perform better with overall less latency compared to AMQP (Sreeraj and Kumar, 2018). In contrast, a recent comparison study found that out of all the common IoT communication protocols, MQTT was scored on the lowest in terms of security (Naik, 2017). This suggests that that SMQTT is the preferred choice for a home security system, as this protocol would meet all the selected design requirements presented in Table 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | MQTT | CoAP | AMQP | RESP |
| Header Size | 2 Byte | 4 Byte | 8 Byte | N/A |
| Architecture | Client/Broker | Client/Server or Client/Broker | Client/Server or Client/Broker | Client/Server |
| Abstraction | Publish/Subscribe | Request/Response or Publish/Subscribe | Request/Response or Publish/Subscribe | Request/Response |
| Message Size | 256mb Max | Small & Undefined | Negotiable & Undefined |  |
| QoS / Reliability | QoS 0 – At most once  QoS 1 – At least once,  QoS 2 – Exactly once | Confirmable or Non-Confirmable Message | Settle Format or Unsettle Format | N/A |
| Transport Protocol | TCP | UDP | TCP, SCTP | TCP |
| Default Port | 1883/8883 (TSL/SSL) | 5683 (UDP) | 5671/5672 (TLS/SSL) | 6379 |

Table 2: Features of most common IoT Application Communication Protocols

## 2.4 Data Exchange Formats

Due to the IoT industry still being in its infancy, there is little standardisation in most of its areas. This holds true for data exchange (Al-Fuqaha et al., 2015), with there being multiple data formats currently used. This section aims to review the current data exchange formats and suggest the most suitable format for the proposed system.

Data serialization is the process of writing the state of an object to a stream and rebuilding this object when received on the other end. This is a way of sending and receiving data over a network. When choosing an appropriate data exchange format, the following must be considered:

* Message Size: The overall number of bytes of the message being sent. With IoT communication protocols being designed for lightweight, small volumes of data, the overall size of the message is a large factor in the communication of IoT devices.
* Serialization Time: This involves translating data structures into the format that can then be used for data transmitting. This data once received by the recipient, can then be reconstructed.
* Deserialization Time: Once received, this is the amount of time taken to reconstruct the data into a format readable by the recipient device or machine.

### 2.4.1 XML

Extensible Markup Language (XML), published in 1996, is one of the oldest data exchange formats created by W3C. It was designed to be used over the internet and remain human legible (Figure 7 shows the XML format). It is text-based, which can be problematic in terms of speed, as it will also need to be parsed character by character.



Figure 7: Code Snippet showing basic XML Schema

### 2.4.2 JSON

JavaScript Object Notation (JSON) is a message format that arose as a subset of the JavaScript programming language. It holds a similar structure to XML, being human-readable with a similar schema, but with less overhead (as shown in Figure 8). In AJAX applications, JSON and XML were compared, with JSON outperforming XML in terms of data size occupancy and transmission speed (Lin et al., 2012).



Figure 8: Code snippet showing basic JSON Schema

### 2.4.3 Protocol Buffers

ProtoBuf is a form of protocol buffer developed by Google which utilises binary instead of text to perform fast serialization and deserialization. When compared to JSON, Protobuf outperformed JSON in both message size and speed in both Java to Java communication and Java to JavaScript, showing promising results (Krebs, 2017). The author claims Protobuf can perform up to six times faster than JSON. It should be noted that this test only tested these environments and should not be used as a whole representative for the two data exchange formats.

### 2.4.4 Comparative Analysis

In a comparison study, the serialization size, average serialization time and average deserialization time were compared between XML, JSON, and ProtoBuf (Sumaray and Makki, 2012). The protocol buffer outperformed both JSON and XML with XML performing the worse in every scenario (shown in Tables 3, 4 and 5). This held true regardless of the size of the object (large or small) and highlights the efficiency of the protocol buffer. Although this study was carried out on a mobile platform, it gives a clear perspective on the performance of each of the data exchange formats available for IoT communication. It can be concluded that the main advantage of using JSON is its human-readable interface, which is not necessary for IoT based systems. Therefore, ProtoBuf is the most suitable due to its performance in the aforementioned qualities.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **XML** | **JSON** | **ProtoBuf** |
| **Book** | 873 | 781 | 687 |
| **Video** | 231 | 139 | 59 |

Table 3: Serialized size in bytes (Sumaray and Makki, 2012)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **XML** | **JSON** | **ProtoBuf** |
| Book | 22.284 | 4.177 | 2.339 |
| Video | 17.884 | 4.097 | 1.800 |

Table 4: Average serialization time in milliseconds (Sumaray and Makki, 2012)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **XML** | **JSON** | **ProtoBuf** |
| **Book** | 7.908 | 1.199 | 0.298 |
| **Video** | 6.742 | 0.755 | 0.197 |

Table 5: Average deserialization time in milliseconds (Sumaray and Makki, 2012)

## 2.5 Security Threats

This section highlights potential security threats relating to IoT devices. The aim of this section is to present the most frequent security vulnerabilities present within the IoT architecture. A 2018 paper highlighted Denial of Service, and Man in the Middle (particularly Address Resolution Protocol Poisoning) being the main cyber threats IoT (Gao et al., 2018). This section will review the relevance of these and other attacks to the proposed system.

### 2.5.1 Denial Of Service Attack

DoS attacks are used to prevent access to devices or network resources, essentially taking the network offline. It is accomplished by flooding a network with packets and redundant requests to the point of the network not being unable to be accessible by authentic users (Prabhakar, 2017). Direct Denial of Service (DDoS) is a more extreme version of the regular DoS, which uses multiple compromised systems instead of a single unit (shown in Figure 8). Infected computers as a collective are called a botnet. A botnet is a collection of computers performing repetitive tasks to keep websites going (Uk.norton.com, 2019). The problem arises when this architecture is used for malicious intent, harvesting a machines power to be used in assisting in DDoS attacks. Figure 9 highlights this, showing an attacker issuing an attack command using the ‘slave’ computers infected by a botnet to floor a victim’s network. Although traditionally used against computers, IoT devices are starting to be the main target, due to their lack of security. In 2017, it was discovered that there were various botnets designed specifically to infect IoT devices (Krebs, 2017), namely ‘Mirai’, ‘Reaper’ and ‘IoTroop’, further adding to the problem. Additionally, nearly half of the ‘Rakos’ botnet collection consisted of Pi models. This suggests that Pi systems are clearly prone to bot nets and precautions must be taken to prevent such attacks.



Figure 9: A DDoS attack being carried out using infected PCs (slaves)

### 2.5.2 Man in The Middle Attack

The MITM has the ability to capture all messages between a server and IoT device via network spoofing. Using this spoofing strategy, it can assume identification of both the IoT device and server and trick both devices into believing they are still communicating with each other (Conti, Dragoni and Lesyk, 2016). Address Resolution Protocol (ARP) Poisoning is a form of MITM attack that uses TCP/IP protocol to acquire physical addresses (MAC) based on the IP addresses of nodes within the network (Abad and Bonilla, 2007). This is done through broadcasting an ARP request on the network, requesting access to the host’s IP address. When this malicious node is on the network, the node being attacked matches its physical address with the IP address of the original communication node. This is done through sending a fraudulent message to the node attacked and recording the error pairing in an APR cache table. The victim node will then communicate with the attacker node by misinterpreting the node as normal communication, sending all data packets towards the malicious node. This can be shown in Figure 10, showing an intercepted connection between a Server and Client. Packet sniffing can be problematic when it comes to sensitive data such as surveillance images in an individual’s home, meaning preventive measures should be prioritised against this attack.

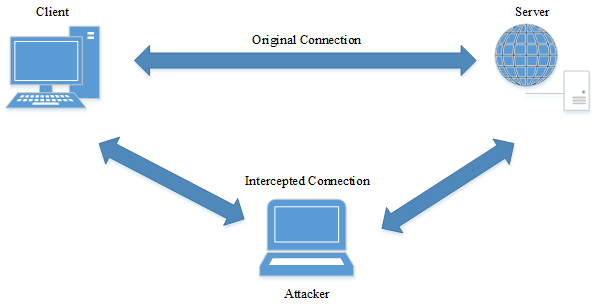


Figure 10: MITM attack showing an intercepted connection from a malicious attacker

### 2.5.3 Brute Force Attacks

Brute force attacks consist of using software to guess as many passwords as possible while attempting to login into a system until the correct password is guessed. This could occur at the device level (attempting to login to the device remotely) and at network level e.g. guessing the passphrase of a Wi-Fi router. WAP2, being the most common form of security for personal Wi-Fi routers, is susceptible to these attacks. Figure 11 shows a client using a malicious script to brute force an insecure website. A 2017 paper concluded that this protocol is especially vulnerable to dictionary-based attacks and key reinstallation attacks (Abo-Soliman and Azer, 2017), which are forms of brute force attacks. Implementing a form of account lockout can be an effective strategy to prevent brute force attacks. The UK government recommends a maximum of 10 login attempts before locking out the account (ukgov, 2015). If remote access to the Pi such as Secure Shell (SSH) is accessible, this allows an attacker to attempt brute force methods on the Pi to attempt to gain access.

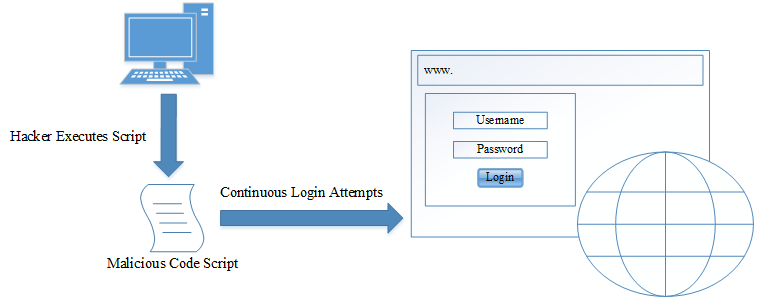


Figure 11: Computer running a malicious script to attempt logins on a website

### 2.5.4 Physical Tampering

Physical manipulation of a device could occur. This is when the device is modified to act in unexpected ways by someone with malicious intent. Physical tampering includes connecting to exposed ports, interrupting a devices power, removing parts or manipulation of camera devices (Elizalde, 2019). It is safe to assume that if this happens the intruder is already in the individual’s home and the home security system has already either captured the intruder and informed the home owner or has been compromised prior to this stage via other approaches. Even though this attack not directly relevant to the proposed system, a way to protect the user’s data if this occurred would be full encryption of the storage and two-factor authentication to prevent users from accessing the micro SD that is present in the device. These encryption approaches will be further examined in the following section.

### 2.5.5 Malicious Code Attack

This is code in any part of the software intended to cause undesirable effects and potentially causing the system to malfunction. This is especially prominent in IoT devices where remote firmware updates are common as it is sometimes impossible or not financially viable to collect these devices for an update (Kvarda et al., 2016). This form of security threat will not occur, as the device will not be updated in such context.

## 2.6 Cryptography Options

This section aims to review the most common forms of cryptography and establish the state of the art methods in securing communication between devices. In accordance with the guidelines set by the UK government (ukgov, 2015)., passwords should not be stored as plain text. The best way to store passwords is established in this section.

### 2.6.1 Symmetric Encryption

Advanced Encryption Standard (AES) is a symmetric form of encryption (Figure 12), supporting key lengths of 127, 192 and 256 bits. In a comparison study against four different algorithms, it was shown that AES has the highest avalanche effect, which is recorded as how much change there is in an output based on the change of an input. Scoring high in the avalanche effect reflects a high level of cryptographic strength. This suggests that, when confidentially and integrity are the highest priority, AES should be used (Patil et al., 2016).

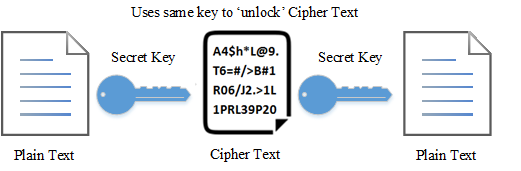


Figure 12: Example of plain text being encrypted using symmetric key encryption

### 2.6.2 Asymmetric Encryption

Rivest–Shamir–Adleman (RSA), named after its creators, works on a public-private key basis that is a form of asymmetric encryption (Figure 13). The public key is available to everyone, i.e. placed on the server (RPi) whereas the private key is used to ‘unlock’ that specific public key. This gives an added layer of security as the data can only be decrypted using the individual’s private key. The private key is usually stored on a client device e.g. computer or mobile phone (Goshwe, 2013). This form of encryption was used successfully in the creation of a payment terminal using a RPi, TSL and AES based encryption, highlighting its efficiency (Kakar, 2016). This can be used in conjunction with SSH to allow remote access to only those users that have the private key in relation to the public key, aiding in the prevention of brute force attacks.

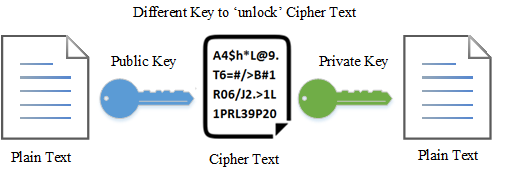


Figure 13: Example of plain text being encrypted using Asymmetric key encryption

### 2.6.3 Hash Encryption

Hashing is a form of one-way cryptography where the text is transformed into a hash algorithm and cannot be reversed. It is used for password storage due to its key feature of non-reversal resulting in not even the owners or administrators being able to see their users’ passwords. Hashing is also used in ensuring message integrity when transferring files. Common hashing algorithms include Message-Digest Algorithm (MD5) and Secure Hash Algorithm (SHA-1). In a comparative study, it was shown that regardless of password length, the SHA-1 algorithm took longer to crack, suggesting this is the superior option when it comes to password hashing (Putri Ratna et al., 2013). The results of the comparison of the two algorithms attack time against a brute force attack for a 6-character password can be seen in Figure 14.



Figure 14: SHA-1 & MD5 brute force successful attack times (Putri Ratna et al., 2013)

### 2.6.4 Salt Encryption

As recommended by the UK government, password hashing alone is not enough (ukgov, 2015). In 2015, a paper demonstrated that regardless of hashing algorithm used, passwords were still susceptible to brute-force, dictionary and rainbow-table attacks (Tatli, 2015), suggesting that without the addition of *salt*, passwords are not safe regardless of the hashing protocol. A *salt* is a random value that is considered cryptographically-strong while being fixed length (Arias, 2018). This *salt* can be added to a hash function, which makes hash functions unique for every input, meaning if two users have the same password, the hash values for each of their passwords would not be the same, as found in traditional solo hashing. This suggests that when hashing passwords, a *salt* should be always added to give an additional layer of security.

## 2.7 Message Integrity

When transporting sensitive data such as private images or messages, message integrity is essential. There are various strategies to identify potential third parties tampering or viewing messages as they are being transported through protocols such as MQTT. This section aims to determine the state of the art strategies in message integrity and ensuring only the authorised individuals and devices are able to receive and access messages. This is usually achieved through a form of ‘digital stamp’ which is added to the payload which can then be used to verify the message was not tamped with by a third party.

### 2.7.1 Hash Algorithms

As mentioned in Section 2.6.3, hashing is an excellent way to store passwords securely. Additionally, hashing can also be used as a form of message integrity. This is achieved by adding the hash algorithm to the payload and verifying it once it reaches a given recipient. The main advantage of this compared to the other mentioned methods is speed. As there is no modification to the network and only to the payload, it has very little impact on message transport speed. The main downside to this form of message integrity is if someone intercepts the message, they could potentially change the message and recalculated the digest of the hash so it would match the recipients hash calculation (Harmoush, 2015).

### 2.7.2 Message Authentication Code Algorithms

MACs add to the security provided by hash algorithms by adding a layer of authentication. Authentication is when the receiver of the message can be ensured that the message originates from a trusted sender. This is done through a form of symmetric key encryption (refer to 2.6.1) meaning only trusted parties have access to the key required to create the digital stamp. The downside to this method is the key must be shared with all entities or receivers (Mehmood, 2017), meaning if the key become compromised by one of the key holders, the whole chain is compromised. In a scenario such as a surveillance camera, all smart phone users with the app connected to the camera would be considered key holders, as well as the surveillance camera.

### 2.7.3 Digital Signatures

Digital signatures improve upon the MAC method of securing MQTT messages by ensuring non-repudiations of the message sent. Non-repudiation is when a messages authorship cannot be disputed and therefore cannot deny it comes from the original recipient (Villanueva, 2015). This utilises asymmetric key encryption, meaning only the sender of the message with access to the private key can generate the digital stamp (Villanueva, 2015). The fundamental difference between this and the former method is that other parties can verify the signature, using the public key, but cannot create the stamp using the private key, which the sender should have. This is relevant is cases such as surveillance as a fraudulent image could be sent by a malicious party that simulates an image being taken by the surveillance camera. With no way to verify if it comes from the correct source, this is a way to remedy this.

### 2.7.4 Comparative Analysis

In their review of MQTT security practices, HiveMQ, the online MQTT cloud broker highlights the use of digital signatures as the highest form of secure practices using the MQTT communication protocol (Hivemq.com, 2015). The main argument being, it addresses the three main issues when it comes to message security: data integrity, authentication and non-repudiation.

## 2.8 Facial Recognition

This section covers the numerous options of face recognition algorithms. As face recognition in generally is computationally expensive, limits are imposed in terms of computational power as the RPi’s capability is limited.

### 2.8.1 Haar Cascade

The Haar Cascade method is a form of object detection algorithm primarily used in face detection. The purpose of the algorithm is to identify objects within a video or image based on feature selection. The algorithm is trained via positive and negative images and once trained, is then used to detect objects in other images. Each image is broken up into ‘Haar-Features’ which can be used to detect specific features of an image, the most common being edge, line and four rectangle features, the latter being used to detect slanted lines (as seen in Figure 15). Each calculated feature results in a single value by summing both rectangles and subtracting the sum of the white rectangle from the sum of the black rectangle. Although possible to calculate every single feature is very computationally expensive (Docs.opencv.org, 2019).

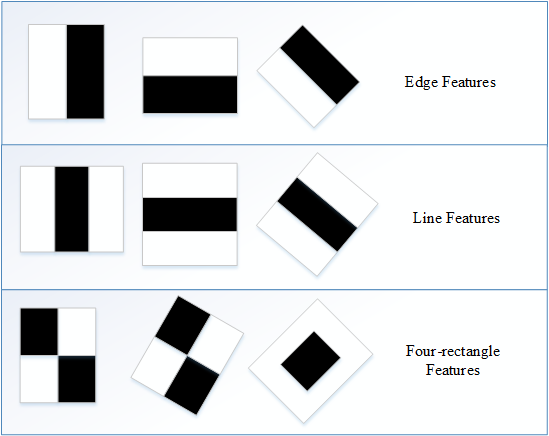


Figure 15: Most commonly detect features in the Haar Cascade algorithm

To remedy this, Adaboost is normally implemented (Viola and Jones, 2001). When it comes to training, every feature is applied on every training image. Adaboost then finds the best threshold for each feature, determining which features contain the minimum error rate. This is done through classification and is continually readjusted until the desired error rate is achieved. The resulting final classifier is a weighted sum of these weak classifiers. The term ‘weak’ is used because these features detected cannot classify an image, but when the selected features are used together, it gives strong classification results.

### 2.8.2 Histogram of Oriented Gradients

HOG is a feature descriptor used to detect objects/faces in the field of computer vision and image processing. The algorithm works by dividing the image into smaller parts via edge detection technique, being highly dependent on the orientations of the edge (Patel and Rajput, 2018). These smaller parts of the image are then further divided into small continuously connected regions called cells.

There are two components of the gradient – horizontal and vertical which can be calculated using Equations 1 and 2:

*Magnitude of gradients = (1)*

*Direction of gradients = (2)*

In their paper (Noman, Yousaf and Velastin, 2016), it is demonstrated that human detection in real time is possible using a simple RPi model in conjunction with the HOG algorithm. Though promising, the author uses foreground estimation, focusing on the area in which humans have a high likelihood of appearing in the frame, to reduce the workload of the Pi. Other strategies such as greyscale conversion and background subtraction techniques were applied, further reducing the computational load on the RPi. It should be highlighted that the Pi model used in the experiment was the model 1b, the original Pi model. This suggests that the aforementioned load reducing techniques may not be necessary on future versions of the RPi, as the one proposed in this thesis. Additionally, the Haar Cascades algorithm and Adaboost classifier was used with a Raspberry Pi 2 model in real time emotion detection, boasting an average accuracy of 94% (Suchitra, Suja P. and Tripathi, 2016). Though impressive, there is no guarantee this has direct carry over to facial recognition. In their 2018 paper (Nagpal et al., 2018), the RPi Zero, the lightweight version of the RPi, was used in conjunction with the Haar-Cascades algorithm for facial recognition. The results concluded a modest 60% accuracy with the classifier, with up to 75% accuracy in ideal conditions. The literature on RPi facial recognition shows no clear consensus on the most accurate and efficient algorithm for the RPi. Further analysis must be carried out to determine the highest performing algorithm with the RPi 3 model.

## 2.9 Security Guidelines for Internet of Things

This section highlights recommended security measures for Internet of Things-based development. In accordance with the best practices outlined by the IoT security foundation (IoT Security Compliance Framework, 2018), the following security measures have been highlighted relating to the project presented in Table 6.

|  |  |  |
| --- | --- | --- |
| Req. No | Requirement | Section |
| 2.4.5.1 | Device has measures to prevent unauthenticated software and files being loaded onto it. | Software |
| 2.4.5.2 | Regarding remote software updates, software images are digitally signed and approved by the appropriate signing authority. | Software |
| 2.4.5.5 | Device disables any virtual ports not required for normal operation, the ports are disabled or only allowed to communicate with authenticated devices | Software |
| 2.4.5.6 | Watchdog timer is present and cannot be disabled to prevent stalling or disruption of operations | Software |
| 2.4.5.8 | Devices protects against unauthorised reversion of software to earlier & potentially insecure version | Software |
| 2.4.5.21 | Devices communicates over TCP, device software uses cryptography to protect data | Software |
| 2.4.6.1 | OS is implemented with relevant security updates prior to release | Process |
| 2.4.6.3 | All unnecessary logins or accounts are disabled at end of software development lifecycle | Software |
| 2.4.6.4 | Minimum access privileges required to function are assigned to relevant files, directories and data | Software |
| 2.4.6.5 | If passwords are stored on device, password files are only accessible by most privileged accounts | Software |
| 2.4.6.6 | All non-essential services have been removed from devices software | Software |
| 2.4.6.7 | All OS command line access to most privileged accounts have been removed from OS | Software |
| 2.4.6.8 | Devices OS Kernel cannot be called by external product level interfaces or unauthorised products | Software |
| 2.4.6.9 | Applications are operated at lowest privilege level and only have access to necessary resources | Software |
| 2.4.6.10 | All supported security features available on OS are enabled | Software |
| 2.4.6.11 | OS is separate from application & is only accessible via defined secure interfaces | Software |
| 2.4.7.1 | Devices prevents unauthorised connections to it, i.e. firewall enabled & internet layer protocol | Software |
| 2.4.7.4 | Device only supports versions of application layer protocols with no known vulnerabilities | Process |
| 2.4.7.5 | If unauthorised change is detected, device should alert admin | Software |
| 2.4.7.6 | All unused ports are closed & only required ports are active | Process |
| 2.4.7.10 | For any Wi-Fi connection using WPA2, secure encryption is used and WPA is disabled | Software |
| 2.4.7.13 | When a TCP protocol such as MQTT is used, it is protected by a TLS connection | Software |
| 2.4.7.18 | Device only enables network, application protocols and network services appropriate for device | Policy |
|  |  |  |
| 2.4.8.4 | Device does not accept null or blank passwords | Software |
| 2.4.8.5 | Device does not allow new passwords containing the users username | Software |
| 2.4.8.6 | Passwords entry follows industry standard | Software |
| 2.4.8.7 | Device has defence against brute force attacks | Software |
| 2.4.8.8 | Device securely stores passwords using industry standard cryptography | Software |
| 2.4.8.11 | Product only allows controlled user account access, no guest user accounts are supported | Software |

Table 6: Relevant IoT best practices in relation to the project

# 3.0 Technology Review

This section details the various technologies proposed to carry out the development and verification of the proposed system. This includes hardware selection, programming language choices, operating system (OS) selection, various tools for security, software libraries and frameworks.

## 3.1 Hardware Selection

The Raspberry Pi Model 3b was selected as the main piece of hardware due to its low cost, operating system malleability and camera support. With support for Linux based operating systems and increased processing power compared to previous models, the board is more than capable of running computer vision libraries for the surveillance system. Border surveillance was carried out successfully using a RPi in a recent paper, demonstrating the boards capabilities. (Abdalla and Veeramanikandasamy, 2017).

## 3.2 Operating System Selection

While there are multiple OS systems available for the Raspberry Pi, Raspbian is currently the recommended, having been designed specifically for the Raspberry Pi (Raspbian.org, 2019). Similarly, when it comes to cyber security, the standard operating system used in cyber security and digital forensics is Kali Linux, derived from the Linux distribution Debian.

### 3.2.1 Raspberry Pi Operating System

Raspbian is an operating system based on the Linux distribution Debian, meaning it has all the features of a standard Linux operating system while being lightweight and designed for the RPi.

### 3.2.2 Cyber Security Operating System

Kali Linux is an operating system contains over 300 programs and tools used in ethnical hacking, making it the golden standard for carrying out simulated attacks. Liang et al., (2016), present three different types of DoS attacks, resulting in all three being successful against a small IoT device using Kali Linux on a separate client. This client was connected to the same network as the IoT device.

## 3.3 Programming Language Selection

This section highlights the justifications for the programming languages selected for development of both the mobile application and surveillance system. Factors such as time constraints, portability, library prominence and compatibility are major considerations in choosing a programming language.

*Raspberry Pi Development*

Python is a programming language which offers tremendous support in terms of Computer Vision, granting access to libraries OpenCV and Dlib (Dlib.net, 2019). The only alternative languages that support these packages are C and C++. Due to time constraints and the authors previous experience with Python, it has been selected for the Raspberry Pi-based development of this thesis.

*Application Development*

Xamarin has been selected as a framework to develop a hybrid mobile application which will receive images and interact with the RPi. Xamarin is based on Visual Studio and the programming language C# and works on a Model-View-ViewModel architecture which allows for clean separation of application logic from its user interface. A recent comparative study showed this specific feature was the biggest advantage over other hybrid and native application approaches (Vishal and Kushwaha, 2018). This, along with the authors previous experience with the programming language C#, makes Xamarin the most suitable choice for the application development.

## 3.4 Library Selection

This section details the various libraries to be used in both the RPi development and Xamarin smart phone application development. As both languages differ greatly, it is important to find the industry standard and more trustworthy packages for each language.

### 3.4.1 Python Library Selection

**OpenCV** is an open source library that offers a range of feature detecting and feature matching algorithms. It is shown that the majority of the algorithms used, detect thousands of features with seconds (Noble, 2016), making it suitable for a face detection system. To highlight the accuracy of this technology, a recent paper showed success in detecting eye fatigue in drivers using OpenCV (Manoharan and Chandrakala, 2015).

**Dlib**is another open source library containing support for deep learning and machine learning respectively. It has excellent image processing support, including SURF, HOG, FHOG algorithms, and additional tools for frontal face detection (Dlib.net, 2019).

**Imutils**is a support library for computer vision, specialising in processing functions such as translation, rotation, and resizing while supporting both Python 2 and 3 respectively (GitHub, 2019). It will be used in conjunction with the computer vision OpenCV to assist in processing images.

**Pickle** is a Python library used for the serialization and deserialization of objects It is used to encode and save objects on a disk in a character stream format (GeeksforGeeks, 2019). This stream contains all the information necessary to reconstruct the original object in a separate python script.

**Face\_Recogniton** is an open source Python library based on Dlib used to recognise and manipulate faces in a simplistic easy to use manner (GitHub, 2019). The library is used for quick face recognition and can be used in live video stream. Being compact and lightweight makes it ideal for a Pi-based surveillance system. Dlib is a prerequisite library for this module and must be installed alongside it.

**PiCamera** is a support package for handling and interacting with the Pi camera. It can be used to show a live feed of the camera on the Pi and capture images which can then be sent using the MQTT protocol.

**Paho MQTT** is a client used for interacting with the MQTT protocol. This will be used to send messages via the Mosquitto broker which can then be received using the M2MQTT client on the smart phone devices.

**Psutil** is a Python library used to get the systems stats during the execution of a Python script. It can record variables such as CPU usage and memory consumption which can be used to analyse the performance of a script for further evaluation.

### 3.4.2 C# Library Selection

**SQLite** is a lightweight version of SQL that can be used within the Xamarin platform. SQLite has the main advantage of being serverless and extremely lightweight with the library being less than 500KiB in size (Sqlite.org, 2019). Being serverless has the advantage of being able to read and write to the database without a network protocol such as TCP/IP. This, along with its lightweight build, make it suitable for smart phone database storage.

**M2Mqtt** is a MQTT client made for all .Net platforms, including support for IoT and M2M communication (GitHub, 2019). Other MQTT clients such as MQTTnet are platform specific and therefore not applicable to Xamarin development

**Xam.Plugins.Notifier** is a small support package for Xamarin Forms used for local notifications. It offers a simplistic way to execute push notifications to both iOS and Android while using the shared code between the two. To enable push notifications, permissions must be granted on iOS in versions higher than 8.0.

**Plugin.Fingerprint** is a small support library for Xamarin supporting both iOS and Android fingerprint authentication. This package would be used in conjunction with a traditional username/password setup as a part of two-factor authentication to grant access to the rest of the application.

**System.Security.Cryptography** is the official cryptography package by Microsoft (Docs.microsoft.com, 2019) offering an array of cryptography algorithms including salt and SHA type hashing. This library will be used to create secure password storage on the application.

## 3.5 Supporting Software Review

Although the development of a secure RPi surveillance system is possible without the assistance of additional software, this section offers additional support software to enhance both the security and system as a whole. Support software such as an advanced firewall is reviewed, and additional software which can be used to speed up the development of the system.

**Visual Studio 2019** **Community** is the latest free version of the IDE released by Microsoft with heavy support for application development (Xamarin) and the programming language C# (Bamburic, 2019). The IDE supports both Android and iPhone emulator interaction and having been made by Microsoft, offers the most support for C#.

**Fail2Ban** is a Python based open source software that interacts with the Pi’s operating system’s firewall (Fail2ban.org, 2019). Its main purpose is to ban IPs that show malicious intent. This includes too many password attempts, seeking for exploits, etc. It is recommended by the Raspberry Pi Foundation, giving it additional credibility (Raspberrypi.org, 2019). The software is also claimed to help protect again DDoS attacks (George, 2019).

**Mosquitto** is an open source message broker which implements the MQTT protocol. Its lightweight size makes it ideal for small low power boards such as the Pi (Eclipse Mosquitto, 2019). As mentioned in the Introduction Section, the aim of this project is to limit the amount of potential security vulnerabilities by removing services such as Cloud vendors from the equation. Mosquitto is an alternative to MQTT cloud vendors such as HiveMQ, which grant public access to their topics and messages.

**Certbot** is a client used for interacting with Let’s Encrypt, which is a company that offer free certificates acting as a Certificate Authority (Letsencrypt.org, 2019). Certbot allows for the automation of Certificate creation and maintenance which can be used in conjunction with the Mosquitto client to ensure message integrity and authentication when communicating through MQTT.

**Xrdp** is an open-source remote desktop client used to give operating systems other than Windows a full functional desktop experience. This is used in conjunction with SSH to grant full remote access to the Pi.

**Android Emulator** will be used to simulate using an Android mobile phone to test and evaluate the functionality of the app before publishing it to the app store. The emulator can simulate everything a traditional Android phone can do as well as allow for testing on multiple API levels.

**DB Browser (SQLite)** is a tool to access SQLite databases. It has the ability to view, change and export data into various different file formats (Sqlitebrowser.org, 2019), most notably the ‘csv’ file format. This will be used to export data from the emulator to be used for further analysis.

# 4.0 System Requirements

This section details the hardware and software requirements of the system. Functional and non-functional requirements are highlighted along with each requirement having a priority level. This is important as criteria with the highest level of priority should always be carried out first, whereas non-functional requirements are additional and should be treated as such.

## 4.1 Hardware and Software Requirements

Hardware requirements (Table 7) are established, with each hardware requirement having an additional backup in case of malfunction. Software requirements are also listed (Table 8) to highlight the relevant programming languages and libraries selected according to the earlier review to achieve the objectives of this thesis.

|  |  |  |
| --- | --- | --- |
| **#** | **Requirement** | **Priority** |
| **1** | Raspberry Pi Model 3b | High |
| **2** | Raspberry Pi NoIR Camera | High |
| **3** | 32Gb Micro SD Card | High |
| **4** | Personal Computer | High |

Table 7: Hardware Requirements for project

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Requirement** | **Justification** | **Priority** |
| **1** | Raspbian Operating System | Optimised operating system made specifically for the Raspberry Pi model | High |
| **2** | Kali Linux Operating System | Operating system used for cybersecurity | High |
| **3** | Python Programming Language | Has industry standard face recognition libraries and MQTT support | High |
| **4** | Python Library - Face\_Recognition | Package used for face recognition modules | High |
| **5** | Python Library - Dlib | Package used for computer vision | High |
| **6** | Python Library – Imutils | Support package for computer vision | Medium |
| **7** | Python Library – Pickle | Package used for object deserialization | Medium |
| **8** | Python Library - OS | Package used to open and save external files | High |
| **9** | Python Library - argparse | Package used to pass arguments into a script when ran | High |
| **10** | Python Library – time | Package used to access the current date and time | Medium |
| **11** | Python Library – pathlib | Prerequisite for protobuf-compiler package | Medium |
| **12** | Python Library – protobuf-compiler | Package used for compiling .proto files | Medium |
| **13** | Python Library - OpenCV | Package used for computer vision | High |
| **14** | C# Library – SQLite | Package used for read/writing to an SQLite database | High |
| **15** | C# Library - M2Mqtt | Package used for creation of MQTT client | High |
| **16** |  |  |  |

Table 8: Software Requirements for project

## 4.2 Functional & Non Functional Requirements

This section presents the functional & non-functional requirements for the development required according to the objectives of this thesis. Development of the functional requirements is carried out first, with those of highest priority completed before other lower priority tasks. Once the core system is completed, all security related requirements are prioritised with the intention of making the system as secure as possible. Table 9 lists the core functionality of the system with Table 10 referring to the non-functional requirements.

|  |  |  |
| --- | --- | --- |
| # | Requirement | Priority |
| 1 | The system allows for photos to be taken from the camera of an individual initially to train the model on recognising their faces | High |
| 2 | The system can encode each image to allow for training to occur | High |
| 3 | The system can use the trained model to identify family members during a live video stream | High |
| 4 | The system can identify individuals that are not recognised as family members during a live video stream | High |
| 5 | The system can capture an image when a person is detected but not identified as a family member | High |
| 6 | The system can label the image with the current date time | Medium |
| 7 | The system can store images into a database | High |
| 8 | The system can securely send the image with relevant details via MQTT | High |
| 9 | All smartphone devices subscribed to the topic can receive the image | High |

Table 9: Core Functional Requirements

|  |  |  |
| --- | --- | --- |
| # | Requirement | Priority |
| 1 | Raspberry Pi should have no open unused ports | High |
| 2 | When SSH port is open, only devices with the RSA private key in relation to the Raspberry Pi RSA public key | High |
| 3 | Both default username and password of the Raspberry Pi are changed, password requirements are in alignment with described in 2.6.3 | High |
| 4 | Access privileges are limited to accounts with SSH access | High |
| 5 | MQTT communication is done securely using Secure MQTT outlined in 2.3.1 | High |
| 6 | Raspberry Pi is encrypted at operating system level | Medium |
| 7 | Database storing image files is encrypted | High |
| 8 | Smartphone users have 2-factor authentication for verification | High |
| 9 | System has protection in place to deal with security threats outlined 2.5 | High |
| 10 | System periodically deletes images on all mobile devices within the smart phone app | Medium |
| 11 | UI of smartphone app is aesthetically pleasing | Low |

Table 10: Non-Functional Requirements

# 5.0 Methodology

The Methodology Section details the process of developing both the surveillance system and the smart phone application. The software development methodology used in the approach is highlighted along with the overall flow of the system. Code fragments are used to explain important parts of the system and how they are developed in terms of actual programming. Communication between the devices is discussed and determined using the knowledge gained from the Literature Review. Finally, security strategies for surveillance, network and application development are shown with the optimal options for security established.

## 5.1 Development Approach

With there being three main areas of criteria with the outlined functional and non-functional requirements, an Incremental Development Lifecycle is proposed. With an incremental approach, each part of the product goes through a design, implementation and testing stage, known as a ‘build’ (shown in Figure 16). This project naturally falls into four parts: RPi (surveillance system) development, application development and application communication protocol development. With this approach, each area of the project can be handled separately and ensured that the system is fully functional before attempting to implement security practices. The following build stages are proposed:

1. Surveillance system development. This includes everything relating to the Pi such as gathering images to train the model, encoding gathered images, training said images on the model and determining the best way to carry out live face recognition. Factors such as power outage are considered and a solution is proposed.
2. Application development. Development of a smartphone app used to communicate with the system remotely and to receive images of possible intruders at the door. This includes handling data storage using SQLite, account creation and login, push notifications and the user interface for MQTT communication.
3. Application communication protocol development. Handling MQTT communication across both the surveillance system and smart phone application is assessed with their respective programming languages and available libraries considered.

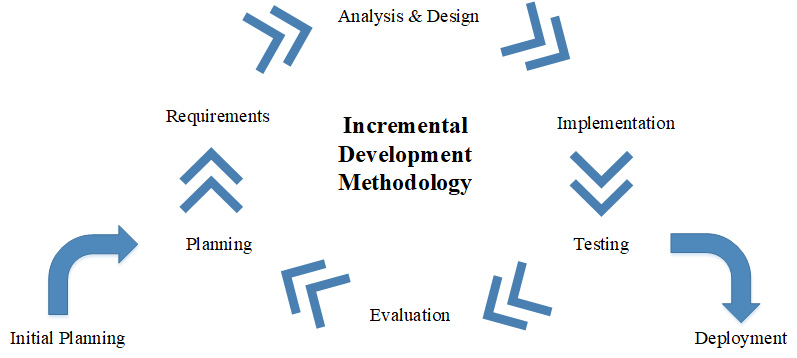


Figure 16: Incremental Software Development Lifecycle for a Home Surveillance System

## 5.2 Main Flow Design & Overview

The main flow of the system is described in Figure 17 and begins by after an unidentified individual enters the premises, prompting the RPi to send the captured image through MQTT. The smart devices connected to the local broker listening on the relevant topic will then receive a push notification which can be pressed to view the image. If the smart phone device is offline, the image will not be received until the user connects to the MQTT protocol. This is saved because the images are sent through QoS 2, which guarantees the payload is sent (as determined in Section 2.3). If the user is not logged in, they will be prompted to login and can then enter their credentials. If their credentials are wrong, they will again be prompted to enter their credentials. Upon success, the user will then be prompted to scan their fingerprint as a form of two-factor authentication. If the user does not have this feature activated or the wrong fingerprint is read, the user will be prompted to take appropriate action. Finally, upon logging in successfully, the user will then be able to view the image and have the option to delete it if requested.

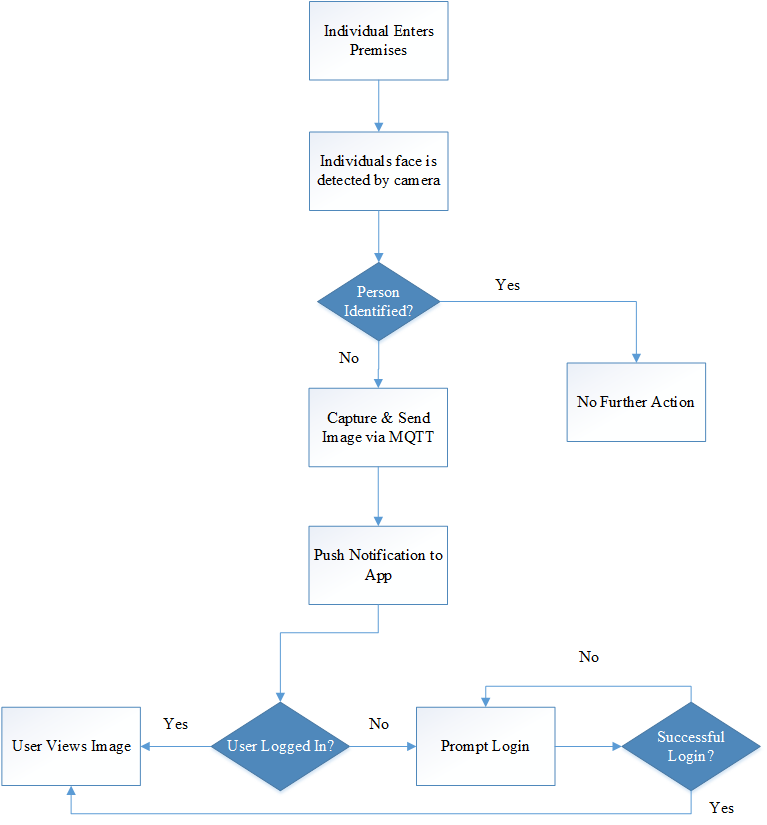


Figure 17: Main Interaction of the Surveillance System and Smart Phone Application

## 5.3 Surveillance System Development

This section details the steps of development relating to the Pi development of the incremental lifecycle. The section is further broken down into smaller components relating to creating a recognisable face dataset and constructing it into a machine-readable format, machine-to-machine communication, carrying out the face recognition in real-time and finally sending the image using MQTT as a mode of transport. The source code for the development of this surveillance system is available on the authors GitHub repository[[1]](#footnote-1).

### 5.3.1 Gathering & Encoding Face Dataset

As mentioned in the Literature Review, face recognition algorithms need to be trained with data (images) in order to recognise an individual’s face. With there being multiple different ways of retrieving images such as using family photos or images already taken, a solution of using the Pi camera has been proposed. The former recommended way of retrieving images can be problematic as there could be multiple people within the photos and the individual must be facing the camera directly to train the frontal view. The script ‘build\_face\_dataset.py’ is ran with arguments passed using the ‘argparse’ library. The argument ‘--output’ is used to set a given location for the images to be placed. For consistency, each individual has their own folder labelled with their given name. For example, passing ‘dataset/Chris’ will store all images captured in the folder relating to the individual named Chris. The script is continually running in a while loop with each stored frame being resized in case of the frame being captured. Every time the user presses the ‘k’ key, an image is captured in ‘.jpg’ format. The loop continues infinitely until the user presses the ‘q’ key to break (see Code Fragment 1 as an example).

1. while True:

2. frame = vs.read()

3. orig = frame.copy()

4. frame = imutils.resize(frame, width=400)

5.

6. rects = detector.detectMultiScale(

7. cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY), scaleFactor=1.1,

8. minNeighbors=5, minSize=(30, 30))

9.

10. for (x, y, w, h) in rects:

11. cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

12.

13. cv2.imshow("Frame", frame)

14. key = cv2.waitKey(1) & 0xFF

15.

16. if key == ord("k"):

17. p = os.path.sep.join([args["output"], "{}.jpg".format(

18. str(total).zfill(5))])

19. cv2.imwrite(p, orig)

20. total += 1

21.

22. elif key == ord("q"):

23. break

Code Fragment 1: Python code handling capturing images using ‘k’ key

For the face images to be read and recognised by a face recognition algorithm, serialization must occur. Serialization is done via the Pickle class which serializes an object before writing it to a file. As this is a one-time process and not done in real-time, a Convolutional Neural Network approach is proposed. This, being a form of deep learning, is the most computationally expensive approach to face recognition, though most accurate (as shown in Section 2.8) and will be run on a separate computer to initiate the face encodings. As images rarely just contain the face of an individual, the face recognition support package is utilised before encoding the images. This library can be used to detect faces in images and crop the image so only the faces are to be encoded beforehand (shown in Figure 18).

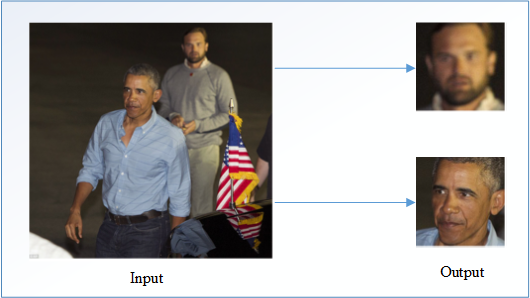


Figure 18: Example of using face\_recognition support library to receive faces

The script ‘encoded\_faces.py’ can then continually run after formatting the images, processing each image into an encoded format and assigning each image to its given folder which was created in the previous step (see Code Fragment 2).

1. for (i, imagePath) in enumerate(imagePaths):

2. print("[INFO] processing image {}/{}".format(i + 1, len(imagePaths)))

3. name = imagePath.split(os.path.sep)[-2]

4.

5. image = cv2.imread(imagePath)

6. rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

7.

8. boxes = face\_recognition.face\_locations(rgb, model=cnn)

9.

10. encodings = face\_recognition.face\_encodings(rgb, boxes)

11.

12. for encoding in encodings:

13. knownEncodings.append(encoding)

14. knownNames.append(name)

15.

Code Fragment 2: Python code showing facial encodings for each individual carried out

### 5.3.2 Real-Time Face Recognition

In the Literature Review, it was concluded there is no clear consensus on the optimal facial recognition algorithm for the RPi. With conflicting results both algorithms are implemented to evaluate the most effective for the RPi. Initially the pickle file ‘encodings.pickle’ is read into the script and the video stream is started. A while loop is created to infinitely loop and prevent the camera from stopping. Each frame is read into a variable with the original frame being stored in a separate variable. The frame of the actual video stream is then modified and resized using the ‘imutils’ support library. The ‘face\_recognition’ class is then reading the frame once it is converted to ‘RGB’ format and the detection method, HOG, is then used to detect if there is a face within the frame (see Code Fragment 3 for an example).

1. while True:

2. frame = vs.read()

3. orig = frame.copy()

4. rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

5. rgb = imutils.resize(frame, width=750)

6. r = frame.shape[1] / float(rgb.shape[1])

7.

8. boxes = face\_recognition.face\_locations(rgb,

9. model=args[“detection\_method])

10. encodings = face\_recognition.face\_encodings(rgb, boxes)

11. names = []

12.

Code Fragment 3: Python code showing frame modification, speeding up real-time processing

After a face has been detected, the script will then cycle through the face encodings created in the previous step, with the name variable being automatically assigned to “Unknown” and overwritten if the frame matches a given encoding. To ensure accuracy, a dictionary is created and added to every time a stored encoded image matches the given frame. The person with the highest count in the dictionaries name is then assigned to the name variable, with the assumption it is them.

1. for encoding in encodings:

2. matches = face\_recognition.compare\_faces(data["encodings"],

3. encoding)

4. name = “Unknown”

5.

6. if True in matches:

7. matchedIdxs = [i for (i, b) in enumerate(matches) if b]

8. counts = {}

9.

10. for i in matchedIdxs:

11. name = data["names"][i]

12. counts[name] = counts.get(name, 0) + 1

13.

14. name = max(counts, key=counts.get)

15.

16. names.append(name)

17.

Code Fragment 4: Python code showing face comparison in real time

After looping through the dataset, there is a box drawn on the live image feed, around the individuals face using the OpenCV module. Text is also drawn on-screen with the default being ‘Unknown’, meaning the individual was not recognised. If there is a match from the previous step, their name is written on the screen instead. When the individual is not recognised, the original frame is captured along with the current date and time used to label the captured image. In terms of image formatting, JPG is the most suitable. Small sacrifices in image quality compared to its PNG format counterpart are a suitable trade-off for its significantly smaller size. Finally, the image path of the taken image is then passed as an argument to the publish script and the script sleeps for 10 seconds to prevent another image of the individual being sent twice.

1. for ((top, right, bottom, left), name) in zip(boxes, names):

2. top = int(top \* r)

3. right = int(right \* r)

4. bottom = int(bottom \* r)

5. left = int(left \* r)

6.

7. cv2.rectangle(frame, (left, top), (right, bottom),(0,255,0),2)

8. y = top - 15 if top - 15 > 15 else top + 15

9. cv2.putText(frame, name, (left, y), cv2.FONT\_HERSHEY\_SIMPLEX,

10. 0.75, (0, 255, 0), 2)

11.

12. if name == 'Unknown':

13. dateTime = datetime.now()

14. p = os.path.sep.join([args["output"], "{}.jpg".format(

15. str(dateTime))])

16. cv2.imwrite(p, orig)

17. subprocess.call(["python3", "/home/pi/Diss/publish.py",

18. "--img", "/home/pi/Diss/dataset/" +

19. str(dateTime) + '.jpg'])

20. time.sleep(10.0)

21.

Code Fragment 5: Python code showing image capture of unidentified subject

### 5.3.3 Power Outage Considerations

If, in the unlikely event that the Pi is forced to be rebooted, measurements are to be taken to ensure the surveillance system remains functional. This can be remedied using a shell script and Crontab, a background process that allows for execution of scripts at set times. The shell script ‘launcher.sh’ was created with the following content:

*sudo python3 surveillance.py*

Once the shell script is created, it must be incorporated into Crontab which can set the script to automatically fire after a reboot using the following command:

*@reboot sh /home/pi/MSc\_Surveillance/launcher.sh >/home/pi/logs/cronlog 2>&1*

This will force the shell script containing instructions to execute ‘surveillance.py’ upon start-up, ensuring the camera remains on at all times when the Raspberry Pi is powered on.

### 5.3.4 Security Implementation

This section aims to deal with security related issues on the RPi in accordance with the IoT Security Foundations guidelines. By fulfilling the relevant requirements set by the foundation, the hypothesis is the device will secure enough to protect against the attacks outlined in Section 2.5. The following areas of security were tackled:

**Password Handling & Access Privileges:** Change both the default username and password. Protocols such as SSH require both a username and password to access a remote device. The default username and password for the Pi is ‘pi’ and ‘raspberry’ for all devices. By changing these we add another layer of security by limiting the amount of information an individual with malicious intent has. Sudo command should require a password. Sudo in Linux operating systems allows commands to be run with admin privileges. By doing this a user must know the password required in order to execute commands at a higher level. This is in accordance with requirement 2.4.5.1 of the IoT Security Guidelines.

**Firewall Protection:** By default, the firewall available on the Raspbian operating system is not configured and must be set up by the user. A firewall is used to enforce a set of rules regarding packet transfer and network communication. Having the ability to block suspicious traffic in an essential countermeasure against network-related attacks such as Denial of Service (see 2.5.1). A recent 2018 paper highlights a rule-based approach (firewall) showing successful mitigation of multiple forms of DDoS attacks against a Raspberry Pi model, suggesting this approach being highly effective (Patel and Upadhyay, 2018).

**WatchDog Implementation:** When dealing with small boards such as the RPi, the possibility of the device crashing is always a possibility. A WatchDog timer allows a device to detect and recover faults and can automate actions such as rebooting the device under the pretence that the device will fail or malfunction (SwitchDoc Labs, 2014). Installing the WatchDog timer can be carried out using the following commands:

*>sudo modprobe bcm2835\_wdt*

*>echo "bcm2835\_wdt" | sudo tee -a /etc/modules*

*>sudo apt-get install watchdog*

*>sudo update-rc.d watchdog defaults*

This is implemented in accordance with Requirement 2.4.5.6 by the IoT Security Foundation.

## 5.4 Application Development

As concluded in the Technology Review section, a hybrid application development approach is taken due to the ease of developments and time constraints on the project. The framework, Xamarin, is used to develop for both the iOS and Android platforms, utilising its Model-View-ViewModel architecture. This section highlights the main functional requirements of the application. The source code for the app is available on the authors GitHub[[2]](#footnote-2).

### 5.4.1 Initial Class Diagram Model

This section attempts to detail the theoretical class diagram to be used for the mobile application. To ensure two-factor authentication, the user must be logged in before being able to access images received via MQTT. Figure 21 highlights the many to many relationships between the images received and the users of the app. For an image to be received, the individual must be logged in, therefore the relationship will always be one to many. A user will have a zero to many relationships with the MQTT image because it can exist without an image being received and have as many as possible. The ‘MQTTMessageTransport’ class has a direct relationship with the ‘MQTTImage’ class as there can only ever be one ‘MQTTImage’ for one ‘MQTTMessageTransport’ class.



Figure 19: UML Class Diagram highlighting the relationships between the user and MQTT interaction

### 5.4.2 Account Creation and Login

To register an account, a user must enter their credentials (email address and password) and confirm their password an additional time to ensure the passwords match. To enforce good security standards, passwords must be at least six characters long and contain a number and special character. Passwords are always hidden from screen to prevent potential eavesdropping. In terms of password storage, passwords will be stored using hash encryption.

As discussed in Section 2.6, password hashing in combination with a *salt* is considered secure password storage and protects against security threats such as brute force attacks (outlined in Section 2.5.3). The SHA521 algorithm was chosen for the hash component as it was determined the most secure algorithm in the Literature Review. To highlight the efficiency of *salt* password hashing, the term ‘password’ was hashed used various different types of hashes discussed in Section 2.6. Table 11 highlights the efficiency of the SHA521 and *salt* combination showing a unique 128-bit length despite the same term being used both times, therefore this combination has been chosen for password storage.

|  |  |  |
| --- | --- | --- |
| Algorithm | Attempt | Result |
| MD5 | 1 | 5F4DCC3B5AA765D61D8327DEB882CF99 |
| MD5 | 2 | 5F4DCC3B5AA765D61D8327DEB882CF99 |
| SHA521 | 1 | B109F3BBBC244EB82441917ED06D618B9008DD09B3BEFD1B5E073  94C706A8BB980B1D7785E5976EC049B46DF5F1326AF5A2EA6D10  3FD07C95385FFAB0CACBC86 |
| SHA521 | 2 | B109F3BBBC244EB82441917ED06D618B9008DD09B3BEFD1B5E073  94C706A8BB980B1D7785E5976EC049B46DF5F1326AF5A2EA6D10  3FD07C95385FFAB0CACBC86 |
| *Salt* + SHA521 | 1 | 4E2589EE5A155A86AC912A5D34755F0E3A7D1F595914373DA638C  20FECD7256EA1647069A2BB48AC421111A875D7F4294C72362925  90302497F84F19E7227D80 |
| *Salt* + SHA521 | 2 | BED4EFA1D4FDBD954BD3705D6A2A78270EC9A52ECFBFB010C618  62AF5C76AF1761FFEB1AEF6ACA1BF5D02B3781AA854FABD2B69C7  90DE74E17ECFEC3CB6AC4BF |

Table 11: Comparison of hashing algorithms using the term 'password'

To achieve this standard of password security, Microsoft’s Cryptography package was used (referenced in Section 3.5) to generate both the hash and the salt result of when a user registers on an app. Code Fragment 7 shows the steps required to generate an SHA521 *salted* hash with both the salt and password being converted into a byte array separately and appended. An additional array is then created for the computed hash using the algorithm. The hashed bytes data is then added to a final array with the original *salt* bytes appended in the end. Finally, the string is converted to Base64 format so it can then be stored within the database.

1. public string ComputeHash(string plainText, byte[] saltBytes)

2. {

3. byte[] plainTextBytes = Encoding.UTF8.GetBytes(plainText);

4. byte[] plainTextWithSaltBytes =

new byte[plainTextBytes.Length + saltBytes.Length];

5.

6. for (int i = 0; i < plainTextBytes.Length; i++)

7. plainTextWithSaltBytes[i] = plainTextBytes[i];

8.

9. for (int i = 0; i < saltBytes.Length; i++)

10. plainTextWithSaltBytes[plainTextBytes.Length + i] =

11. saltBytes[i];

12.

13. HashAlgorithm sha521 = new SHA512Managed();

14. byte[] hashBytes=sha521.ComputeHash(plainTextWithSaltBytes)

15.

16. for (int i = 0; i < hashBytes.Length; i++)

17. hashWithSaltBytes[i] = hashBytes[i];

18.

19. string hashValue=Convert.ToBase64String(hashWithSaltBytes);

20. return hashValue;

21. }

Code Fragment 6: C# Pseudocode showing password encryption with SHA512 + Salt

For a user to login, the text entered within the password field must be checked against the stored password for that particular user. Code Fragment 8 highlights the method ‘VerifyHash’ which takes in the attempted password as parameter and the corresponding hashed password for that particular user. The hash is first converted from Base64 back into a byte array and creates a new hash algorithm using the SHA512 algorithm. As a precaution, the *salted* hash is checked against the length in bytes of the algorithm, returning false if the lengths do not match. The original *salt* byte data can then be separated by subtracting the length of the full *salted* hash by the hash size in bytes. Finally, the *salt* from the end of the hash is copied to the new array and the hash is computed using the plain text and *salt* value, allowing for a successful login if the result matched the original password.

1. public bool VerifyHash(string plainText, string hashValue)

2. {

3. byte[] hashWithSaltBytes = Convert.FromBase64String(hashValue);

4. int hashSizeInBits, hashSizeInBytes;

5. HashAlgorithm sha521 = new SHA512Managed();

6. hashSizeInBits = 512;

7. hashSizeInBytes = hashSizeInBits / 8;

8.

9. if (hashWithSaltBytes.Length < hashSizeInBytes)

10. return false;

11.

12. byte[] saltBytes = new byte[hashWithSaltBytes.Length -

13. hashSizeInBytes];

14.

15. for (int i = 0; i < saltBytes.Length; i++)

16. saltBytes[i] = hashWithSaltBytes[hashSizeInBytes + i];

17.

18. string expectedHashString = ComputeHash(plainText, saltBytes);

19. return (hashValue == expectedHashString);

20. }

Code Fragment 7: C# Code showing login validation by comparing two hash values

After a successful login, fingerprint authentication is then used as the second component of two-factor authentication (shown in Figure 20). As a preventive measure, the user must have a registered fingerprint to access the app, to prevent bypassing this step of the authentication process. Once a user successfully validates their fingerprint, the user is now granted access to the surveillance images and can connect to the MQTT broker.

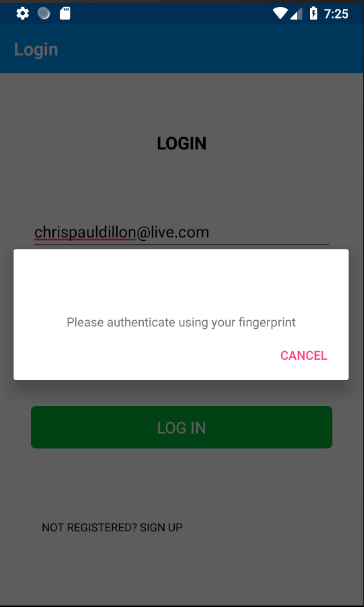


Figure 20: Android Emulator showing fingerprint verification

### 5.4.3 Database Development

For the images to be stored and viewed by the user, SQLite has been proposed to handle MQTT images and users of the app. Figure 20 highlights the relationship between the user and the MQTT protocol. The entity ‘MQTTObject’ shares a ‘one to many relationship’ with ‘MQTTImage’ as an image will always be created when the RPi sends out an MQTT message. Additionally, as there are multiple users subscribed to the same topic, each of them will receive an image therefore creating an ‘MQTTImage’ unique to them. Users can exist without receiving an image and therefore share a ‘zero to many relationship’ with images sent from the surveillance camera. They can also obtain an infinite amount of images from the camera, which is represented by the ‘many’ component of this relationship.

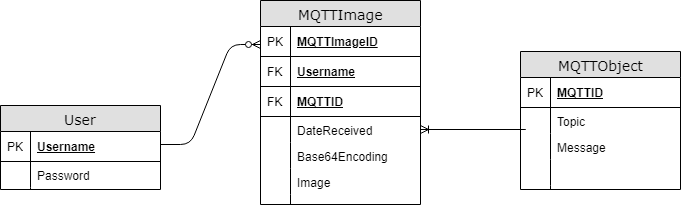


Figure 21: Entity Relationship showing overall database structure

### 5.4.4 Handling Received Images

Push notification permissions must be enabled in both iOS and Android for the user to be notified when an image is sent through MQTT. This push notification will notify the user of someone being at his or her door and once pressed will show the image of the potential intruder on screen. As displaying the images sent from the Pi model is the core functionality of the application, this is highly prioritised in terms of user interface design. For security purposes, the images within the application can only be shown while the user is logged in. The user can then view all recent images within a list view and select an image to allow for viewing the image on a larger scale.

Upon receiving an image via MQTT, the image is added to the image database and is stored in base64 format. This base64 code can then be reconstructed into an image every time an ‘MQTTImage’ object is created (as shown in Code Fragment 10).

1. public async void AddImageToDB(string base64)

2. {

3. await App.Database.SaveImageAsync(new MQTTImageReceived

4. {

5. dateTime = DateTime.Now,

6. ImageBase64 = base64

7. });

8. }

9.

10. private void MqttMessageTransportMessageReceived(MqttMessageTransport obj)

11. {

12. AddImageToDB(obj.Message);

13. }

Code Fragment 8: C# Pseudocode on handling messages received from MQTT

### 5.4.5 MQTT Configuration Settings

For testing purposes and allowing the user to have more control over the communication protocol, a settings page (shown in Figure 22) is implemented allowing for the three levels of security (detailed in Section 5.5.3). The page allows a user to load certificate files and enable tls for the maximum amount of security possible in MQTT communication. Having input a username and password, as previously mentioned, will create a client with username and password based authentication.

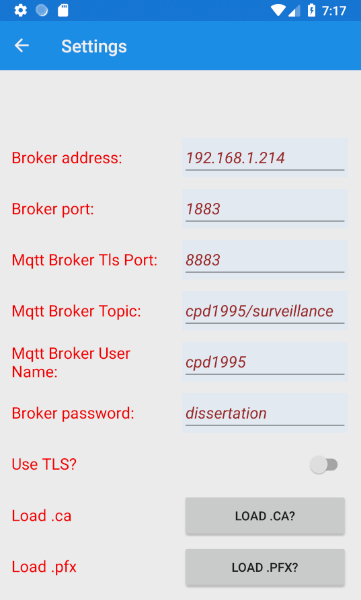


Figure 22: Android Emulator showing MQTT configuration settings page

## 5.5 MQTT Communication

This section describes the strategies and development used to establish communication between the surveillance system and the smart phone applications. As discussed in Section 2.2, MQTT has been determined as the most suitable and reliable approach to communication between the app and the Pi. Cloud MQTT vendors have been ruled out in this experiment, to limit the scope of potential vulnerabilities. Mosquitto is selected as the broker, as it can be hosted on the RPi. The broker (server) interacts with the clients to send out messages on various topics (seen in Figure 21). As two different development languages are used, the Paho MQTT client has been selected for the Pi (using Python) and the M2Mqtt client has been selected for the smart devices (using C#). QoS level 2 was chosen as determined by the Literature Review due to its ability to retain messages, meaning if the user is not connected at the time a message is sent, it will be retained for when they activate MQTT again.

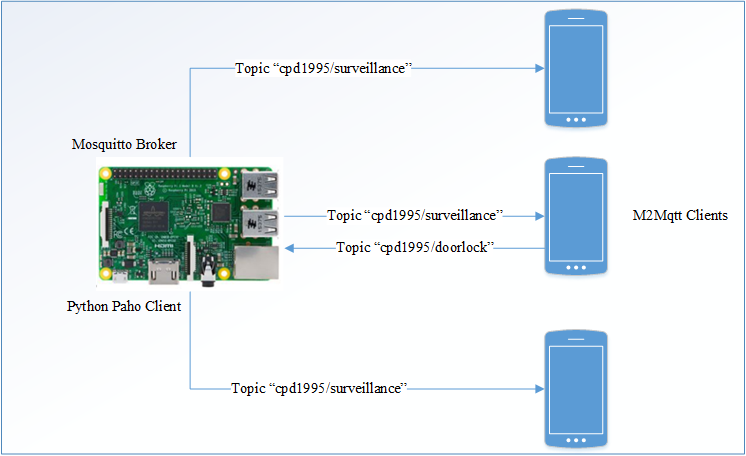


Figure 23: Mosquitto broker (RPi) interacting with Paho & M2Mqtt clients

### 5.5.1 Surveillance System Communication

The script ‘publish.py’ has been set up to be executed using the previous script ‘surveillance.py’ once an unknown face has been detected. The argument took when the script is executed is the path to the newly created image which is then converted into Base64 encoding to be sent via MQTT under the topic ‘cpd1995/surveillance’ to all connected smartphone devices. Code Fragment 9 shows the logic used to carry out this action, with this script being fired every time a potential intruder has been detected.

1. def publishEncodedImage():

2. with open(filename, "rb") as image\_file:

3. encoded = base64.b64encode(image\_file.read())

4. client.publish(topic,encoded,qos)

5.

6. client= paho.Client("smartapp")

7. client.tls\_set('/home/pi/ca.crt')

8. client.connect(broker, 8883)

9. publishEncodedImage()

Code Fragment 9: Python code showing publish method for Paho Client

### 5.5.2 Application MQTT Communication

As discussed in Section 2.8, there are various levels of security and message integrity surrounding the transporting of MQTT messages. To offer various levels of secure MQTT messaging, the app was configured to automatically detect which level of authentication the user requires based on the volume of information supplied. Diagram 24 shows the app successfully connecting to the Mosquitto broker with no security, username and password authentication and certificate-based authentication, prompting a success message once connected.

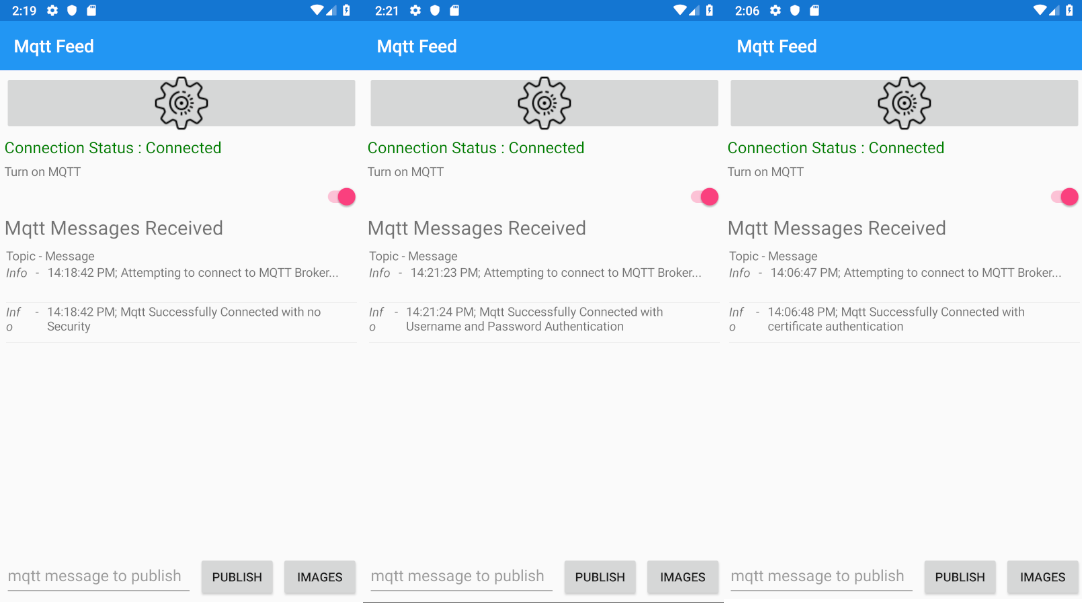


Figure 24: App showing successful connection on three different security layers

For this to be achieved, the app detects what level of authentication the user wishes to use using the settings page (shown in Figure 22). Code Fragment 10 shows the logic used to carry out this task, resulting in successful connection to the MQTT broker on multiple security layers. The app will always check and prioritise the higher level of security selected, meaning if the user selects ‘tls’ then this will be prioritised over username and password authentication and the former being prioritised over no security. The client is always configured for QoS level 2, giving the advantage of full message rention in the case of a message is sent while the client is inactive.

1. private MqttClient \_client;

2. private IMQTTSettings \_mqttSetting;

3.

4. public async Task InitializeMQTT()

5. {

6. if (\_mqttSetting.UseTls)

7. {

8. \_client = new MqttClient(

9. \_mqttSetting.MqttBrokerAddress,

10. Int32.Parse(\_xpdSetting.MqttBrokerTlsPort),

11. \_mqttSetting.UseTls,

12. MqttSslProtocols.TLSv1\_2,

13. new RemoteCertificateValidationCallback(

14. MyRemoteCertificateValidationCallback),

15. new LocalCertificateSelectionCallback(

16. MyLocalCertificateSelectionCallback});

17. }

18. else if(\_mqttSetting.MqttUsername != “”)

19. {

20. \_client = new MqttClient(\_mqttSetting.MqttBrokerAddress,

21. \_mqttSetting.MqttUsername, \_mqttSetting.MqttPassword);

22. }

23. else

24. {

25. \_client = new MqttClient(\_xpdSetting.MqttBrokerAddress);

26. }

27. \_client.MqttMsgPublishReceived+=\_client\_MqttMsgPublishReceived;

28. \_client.Subscribe(new String[] { \_xpdSetting.MqttBrokerTopic },

29. new byte[] { MqttMsgBase.QOS\_LEVEL\_EXACTLY\_ONCE });

30. \_client.ConnectionClosed += \_client\_ConnectionClosed;

31. \_client.Connect(clientId, null, null, false, 60);

32. }

Code Fragment 10: C# Code handling different levels of secure MQTT M2Mqtt client creation

### 5.5.3 MQTT Communication Security

Mosquitto offers three ways to verify client authentication, being client id, username and password and client certificates, each providing additional security to the former.

**Client ID Prefix:** Every client that connects to the broker must supply a unique client id to establish connection. A client id prefix sets a restriction on the name of each client id that a connection is permitted. For example, in the Mosquitto configuration file, the variable ‘clientid\_prefixes’ could be assigned a value of ‘CPD’, meaning each client that connects to the broker must have this prefix at the beginning of their client id. This is the simplest form of client authentication, which could easily be broken if an individual becomes aware of the prefix, as there is no form of non-repudiation in this method.

**Username and Password:** Username and password authentication goes a step further, limiting connections to those who are in possession of said username and password. This, while additionally more secure than the client prefix method, is not secure. Authentication in this manner faces the same issues as the former, with a lack of non-repudiation and the fact that both the username and password is transmitted in plain text. Setting the variables ‘allow\_anonymous’ to false and ‘password\_file’ to route to the file with the password in the Mosquitto configuration file allows for this authentication to be possible. This means connections are not permitted without a valid username or password.

**Payload Encryption:** Payload encryption can be used in addition or as an alternative to certificate based encryption, having the advantage of the data being encrypted end-to-end and not just between the broker and client. Both symmetric and asymmetric key encryption can be used as this is carried out independently of the broker and client relationship. If symmetric key encryption and a hash algorithm were to be selected, this would match the level of message integrity outlined in Section 2.7.2, meeting the data integrity and authentication criteria in secure message transport. Additionally, asymmetric key encryption could be used, though this would not reach a state of non-repudiation as there is no third party such as a Certificate Authority to verify the sender’s identity.

**Certificate-based Encryption**: As discussed in the Literature Review section, the ideal form of MQTT security is using Certificates. OpenSSL has been selected to create certificate along with RSA encryption to assist in privacy. This can then be used to generate a certificate which will be used to encrypt and authenticate the communication between the clients and the broker. This form of encryption uses TLS (Transport Layer Security Protocol) to secure the communication between the devices. As shown in Figure 21, the Certificate Authority uses a certification file (.crt) to establish communication between the client and the server. The server uses the public key and certificate to communicate with the client via TLS. Once the handshake has been confirmed by the Certificate Authority, the client can then send an encrypted message using the public key (available to anyone). The message can then only be decrypted using the corresponding private key and the message can then be read. This method addresses the three main challenges when it comes to network communication: authentication, data integrity, and non-repudiation. Data integrity is assured because of the RSA asymmetric encryption. Authentication is verified using the Certificate Authority along with non-repudiation because of the private key, meaning only the individual with the key can decrypt and sign messages.

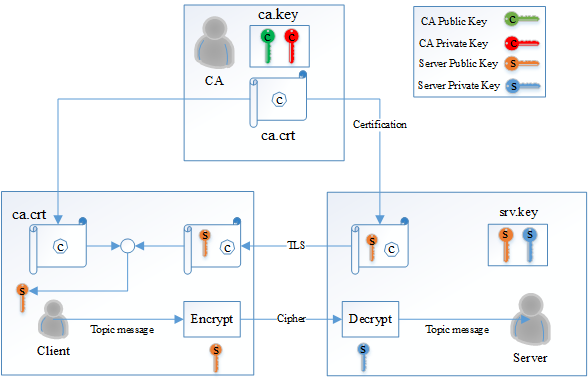


Figure 25: MQTT Communication secured using Certification and TLS

# 6.0 Experiment Setup

## 6.1 System Testbed

### 6.1.1 Latency

When testing the latency of the system, it was important to consider the ‘convenience to security’ ratio, meaning at what point does the inconvenience of higher levels of security hinder the users experience. Three levels of MQTT security were tested: unsecured, username and password authentication and certificate-based authentication. For the surveillance system, a modified script of ‘publish.py’ was created to log the current time in milliseconds each time an image was published. Code Fragment 11 details the code used to collect data over an unsecured MQTT connection. This would continually loop until reaching the maximum number of iterations (2000) and append the data to the CSV file. On the app, as the date and time of each image received was already recorded, the SQLite database file could then be exported as a CSV file and the results were then joined and synchronised using the Python library, Pandas.

1. broker="localhost"

2. filename=args["img"]

3. topic="cpd1995/surveillance"

4. qos=2

5. csvData = [['TimeStamp']]

6.

7. def publishEncodedImage():

8. with open(filename, "rb") as image\_file:

9. encoded = base64.b64encode(image\_file.read())

10. client.publish(topic,encoded,qos)

11. counter = counter + 1

12.

13. if counter >= 2000:

14. break

15.

16. time.sleep(3.0)

17.

18. client= paho.Client("smartapp")

19. client.connect(broker, 1883)

20. publishEncodedImage()

21. my\_df = pd.DataFrame(csvData)

22. my\_df.to\_csv(‘Results.csv', index=False, header=False)

Code Fragment 11: Modified version of script shown in Code Fragment 6

### 6.1.2 System Accuracy & Performance Setup

To evaluate the accuracy of the system, the script ‘surveillance.py’ was modified to record additional variables each time the system recognised a face. Code Fragment 12 shows every face recognition detection being recorded as either true or negative as well as the CPU usage and memory usage being recorded. The results are then appended to a CSV file.

1. counter = 0

2. csvData = [['TimeStamp', 'DetectedFace','CPU Usage', 'Memory']]

3.

4. for ((top, right, bottom, left), name) in zip(boxes, names):

5. top = int(top \* r)

6. right = int(right \* r)

7. bottom = int(bottom \* r)

8. left = int(left \* r)

9.

10. cv2.rectangle(frame, (left, top), (right, bottom),(0,255,0)

11. y = top - 15 if top - 15 > 15 else top + 15

12. cv2.putText(frame, name, (left, y), cv2.FONT\_HERSHEY\_SIMPLEX,

13. 0.75, (0, 255, 0), 2)

14.

15. if name == 'Unknown':

16. dateTime = datetime.now()

17. csvData.append([str(dateTime), 'False',

18. psutil.cpu\_percent(), psutil.virtual\_memory()])

19. counter = counter + 1

20. else:

21. dateTime = dateTime.now()

22. csvData.append([str(dateTime), 'True',

23. psutil.cpu\_percent(), psutil.virtual\_memory()])

24. counter = counter + 1

25.

26. if counter > 100:

27. break

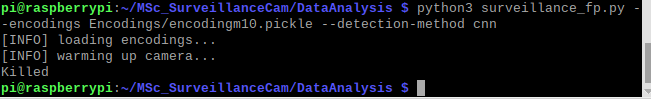
28.

29. my\_df = pd.DataFrame(csvData)

30. my\_df.to\_csv(‘Accuracy.csv', index=False, header=False)

Code Fragment 12: Modified version of 'surveillance.py' script used to record accuracy

This script was tested using both the Haar-Cascades and HOG algorithm as detailed in Section 2.8. Upon attempting to run the script with a Deep Learning algorithm such as the Convolutional Neural Network (CNN), the script was automatically killed off. This is due to the high levels of memory consumption and CPU usage when using such method. Therefore, the CNN is not used in the experiment.



## 6.3 Cyber Security Testbed Setup

As mentioned in the Technology Review section, Kali Linux was selected to test the system. It was established in Section 2.5 that DoS, MitM and Brute Force Attacks were the biggest threats to IoT devices. This section offers a way to test the strength of the security measures put in place to defend against such attacks.

### 6.3.1 Dictionary Attacks

Dictionary attacks, a form of brute forcing, can be used to crack passwords by using a word list and executing a command based on a specific IP address. In this experiment, the IP address of the RPi is 192.168.1.214. Hydra is a piece of software built into the Kali Linux OS and can be used to test the strength of protocols such as SSH. As this program accepts a word list, which is used to guess the password of the victim, ‘rockyou’ has been selected which is available on the OS by default.

>hydra -s 22 -l pi -P /usr/share/wordlists/rockyou.txt 192.168.1.214 -t 4 ssh

The software continually guesses the password by attempting to login using the list of stored passwords in the given word list. Figure 26 shows the continual login attempts to SSH using the Hydra software package.

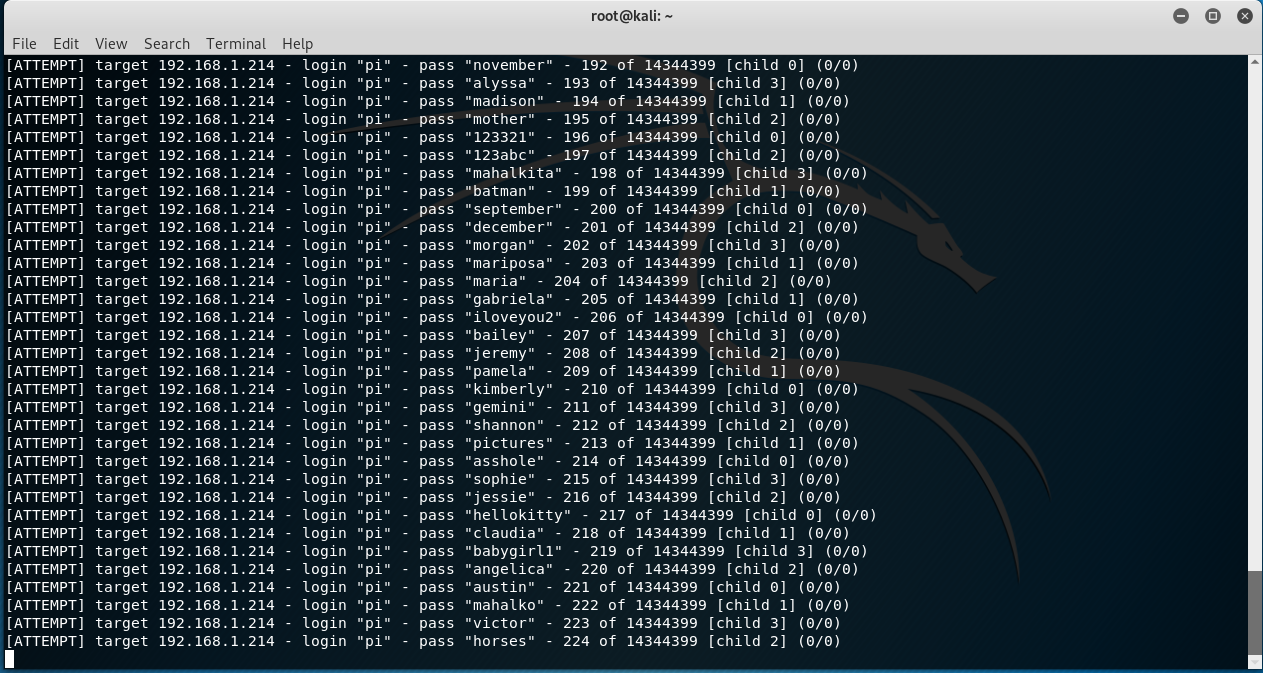


Figure 26: Kali Linux Kernel showing Dictionary Attack using 'Hydra' Software

# 7.0 Evaluation

This section is used to highlight the results of the project and determine the overall success. Data analysis is performed in terms of algorithm accuracy, CPU stress, system latency (between the smartphone devices and the RPi), portability and determining how secure the model is against the attacks discussed in Section 2.5. Jupyter Notebook was used for data analysis and the file can be found on the authors GitHub[[3]](#footnote-3).

## 7.1 Surveillance System Performance

### 7.1.1 Accuracy of Algorithms

To evaluate the accuracy of the system, multiple tests have been setup to determine the volume of correct classifications relative to the overall number of attempts the system used to classify the subject. Table 12 offers four possible outcomes for the experiment which are then used as measures to give an overall picture of the accuracy of the system.

|  |  |
| --- | --- |
| Outcome | Scenario |
| True Positive (TP) | Model predicts author is in camera view and he actually is |
| True Negative (TN) | Model predicts author is not in camera view and someone else is in camera view |
| False Positive (FP) | Model predicts author is in camera view but someone else is in camera view |
| False Negative (FN) | Model predicts author is not in camera view but he actually is |

Table 12: Performance matrix table giving four possible outcomes

As determined by the Literature Review, there is no clear consensus on the best face recognition algorithm for small boards for the RPi, with both the HOG and Haar-Cascades methods being frequently used. Both the Histogram of Gradients and Haar-Cascades algorithms are evaluated.

To get a deeper understanding of the efficiency of the given algorithms, this experiment was split into three parts based on the volume of images used to train the classifier of that specific individual. One, five and ten images were used in the experiment, with the hypothesis of the greater number of images trained on an individual, the more accurate the classifier.

The results shown in Table 13 and Table 14 show a clear disparity between the number of TP results versus the number of TN results. It can be safely said that both classifiers are incredibly accurate at distinguishing between faces. However, both classifiers have a lower success rate when it comes to accurately detecting an individual’s face when the model has been trained on that individual, regardless of number of images trained. To summarise, this means that both classifiers are more prone to not detecting an individual trained on the model and incorrectly mislabelling them as unknown. Both classifiers are highly accurate in not mislabelling an individual as another.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Images Encoded | TP | TN | FP | FN | Sample Size |
| 1 | 68 | 98 | 2 | 32 | 200 |
| 5 | 78 | 97 | 3 | 22 | 200 |
| 10 | 77 | 99 | 1 | 23 | 200 |

Table 13: HOG Algorithm Confusion Matrix using different volumes of encoded images

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Images Encoded | TP | TN | FN | FN | Sample Size |
| 1 | 50 | 96 | 4 | 50 | 200 |
| 5 | 55 | 95 | 5 | 45 | 200 |
| 10 | 59 | 95 | 5 | 50 | 200 |

Table 14: Haar-Cascades Algorithm Confusion Matrix using different volumes of encoded images

To determine the true accuracy of the classifier, the amount of correct classifications must be compared to the entire number of samples. The following equation achieves this goal using the results gained:

(3)

Table 15 highlights the accuracy differential between the algorithms with a trend of encoding more images appeared to improve classifier accuracy across both algorithms, with diminishing returns after five images encoded. The HOG classifier appears to outperform the Haar-Cascades algorithm across the board boasting a peak accuracy of 88%.

|  |  |  |
| --- | --- | --- |
| Algorithm | Images Encoded | Accuracy |
| HOG | 1 | 83% |
| HOG | 5 | 88% |
| HOG | 10 | 88% |
| Haar-Cascades | 1 | 73% |
| Haar-Cascades | 5 | 75% |
| Haar-Cascades | 10 | 77% |

Table 15: Accuracy of algorithms using different volumes of images encoded

### 7.1.2 Computational Stress

When it comes to small IoT devices such as the Pi, computational stress must be a factor in the development of a system that needs to run indefinitely such that of a surveillance system. As previously discussed, Deep Learning algorithms such as the Convolutional Neural Network cannot be run on a device with limited computational power and no graphics card. Two algorithms were reviewed for their suitability, one being the Haar-Cascades algorithm and the HOG method. To make the experiment fair, the system ran as it would in production with no additional software running that could skew the results for both the experiments. As shown in Figure 28, there is quite a remarkable difference in CPU stress between the two algorithms, with the HOG method of face recognition consistently being less stressful on the CPU compared to the Haar-Cascades algorithm. These results were consistent regardless of the volume of images used.



Figure 27: Difference in Computational Stress between the HOG and Haar algorithms

### 7.1.3 Memory Consumption

To further evaluate the performance of the algorithm, memory consumption was used as an additional variable. As shown in Figure 29, it is clear that the Haar-Cascades algorithm consumes significantly more memory than the HOG algorithm, with it consistently consuming more memory.



Figure 28: Difference in Memory Consumption between the HOG and Haar algorithms using 10 encoded images

Table 16 further highlights the performance difference between the two algorithms with HOG out performing Haar regardless of the volume of images used in terms of both CPU stress and Memory usage. It can be concluded with these results and the results of the accuracy comparison that Histogram of Gradients is the superior algorithm for this experiment. While the literature suggests that a Deep Learning approach is potentially superior, the results clearly show that the HOG algorithm is more than suitable for a face recognition surveillance system.

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Images Encoded | CPU Usage % (Mean) | Memory Usage % (Mean) |
| Haar | 1 | 44% | 52% |
| Haar | 5 | 40% | 69% |
| Haar | 10 | 76% | 76% |
| HOG | 1 | 26% | 72% |
| HOG | 5 | 25% | 47% |
| HOG | 10 | 33% | 67% |

Table 16: Comparison of CPU & Memory usage between the HOG & Haar algorithms

### 7.1.4 System Latency

Latency is an important variable in the development of a surveillance system as images must be received as quickly as possible to the recipients. To test for latency, sample images were sent every minute and the time between the sending of the image and receiving it on the smart device determines the latency. Figure 29 shows the disparity between the three levels of authentication used, showing a clear increase in latency when it comes to certificate-based communication. These results shown there is little difference when it comes to using the security options the MQTT protocol offers.

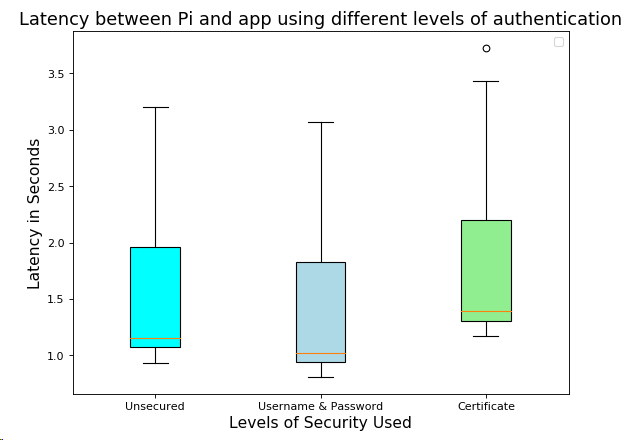


Figure : Boxplot of Latency using three different levels of authentication

## 7.2 App Portability

During the development component of this thesis, the author did not have access to a Mac or MacBook, resulting in there being no option to test the iOS version of the app. To evaluate the portability of the app, the app once completed, was attempted to be ran using a MacBook. The app was initially attempted to be cloned to Xcode, Apple’s iOS platform, and was unable to be cloned or ran on Xcode v9.3, meaning the native platform was unable to run the app. Visual Studio for Mac was then installed, without the Android emulator counterpart to further test the portability. Figure 29 shows the list of errors and warnings that occurred when trying to run the app on this platform, resulting on the app still unable to be launched. The app was unable to be ran on the current version of Xcode previously mentioned, to remedy this, the author attempted to upgrade to the latest version of Xcode, but required an Apple developer account in order to do so.

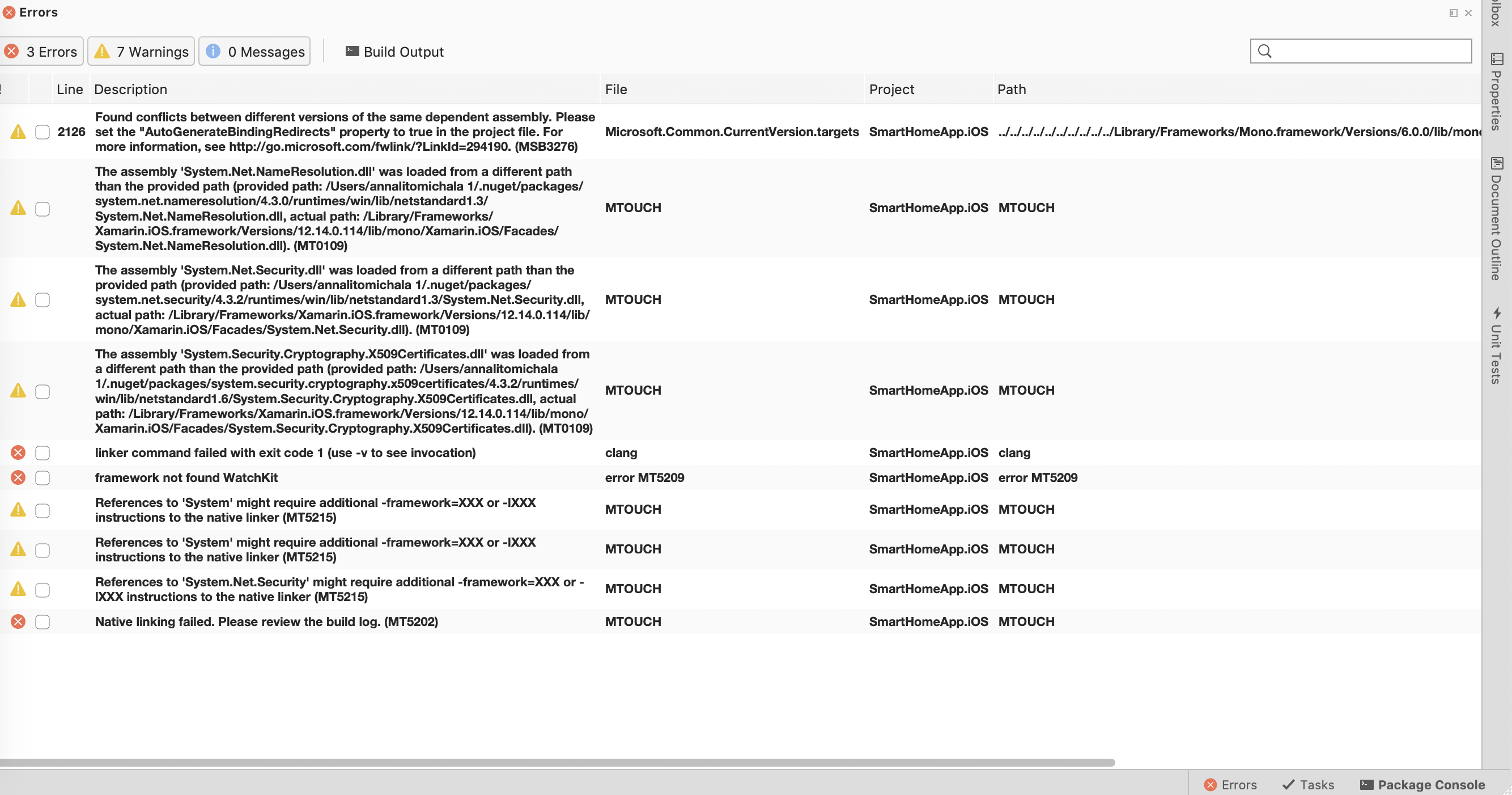


Figure 30: List of errors when attempting to run app on Visual Studio for Mac

## 7.3 Security

### 7.3.1 Brute Force

When it comes to protecting against brute force strategies such as dictionary attacks there are multiple methods available. The first being the most trivial, simply changing the username and password. With changing the username, it makes attacks such as these indefinitely more difficult as the dictionary attack will be forced to guess both the username and password at the same time, granting another layer of security that needs to be broken. To test the viability of strong password practices, the following passwords were selected for the login for the RPi:

Weak Password – animal

Strong Password – face4%3£l1nk3d

Additional strategies such as firewall protection are also included in the experiment as it was determined there is firewall software used to specifically defend against cyber security attacks (as shown in Section 3.5). Finally, as recommended by the IoT Security Foundation, disabling SSH can also protect against these types of attacks.

Table 17 shows the various security measures taken for each scenario and whether they were successful in protecting against an attack or not. It can be safely concluded that having a strong password, firewall protection or disabling SSH are all effective ways to prevent against brute forcing. The results also highlight the dangers of having a weak password as commonly used passwords are used in word lists and are easily guessed by these type of attacks. When it comes to selecting a strategy for protection against dictionary attacks, it is recommended both a firewall and strong password are used if the SSH is absolutely necessary. If SSH is not required to be open, then the safest option is closing the port.

|  |  |  |
| --- | --- | --- |
| Security Measures | No. of Login Attempts | Password Cracked |
| Weak Password | 884 | Successful |
| Strong Password | 52408 | Unsuccessful |
| Firewall Protection (Fail2Ban) | 0 | Unsuccessful |
| Disabled SSH (Closed Ports) | 0 | Unsuccessful |

Table 17: Results of security measures to prevent against a Brute Force attack

### 7.3.2 Denial Of Service

### 7.3.3 Man In The Middle

### 7.3.4 IoT Security Foundation Guidelines

In accordance with the best practices outlined by the IoT security foundation (IoT Security Compliance Framework, 2018), the following security measures have been highlighted relating to the project:

|  |  |
| --- | --- |
| *Req. No* | *Solution* |
| 2.4.5.1 | Device has restricted access privileges with administrative commands such as ‘sudo’ being unable to be used with a password (refer to guidelines outline in Section 5.5.1) |
| 2.4.5.2 | There is no remote software upgrades to the system |
| 2.4.5.5 | The only open ports barring port 8883, which certificate authentication is required to access |
| 2.4.5.6 | A WatchDog timer has been implemented. Refer to Section 5.5.1 for additional details |
| 2.4.5.8 |  |
| 2.4.5.21 | Device communicates over MQTT. Refer to section 2.3.1 for additional details. |
| 2.4.6.1 | The command ‘sudo apt-get update && apt-get upgrade’ ensures the device has the latest security fixes |
| 2.4.6.3 | No additional accounts were created. |
| 2.4.6.4 | Limitations on access privileges have been carried out. Refer to Section 5.5.1 for additional details |
| 2.4.6.5 | Passwords are not stored on the device. |
| 2.4.6.6 | The operating system Raspbian was installed with only the software required to function |
| 2.4.6.7 | Device has restricted access privileges with administrative commands such as ‘sudo’ being unable to be used with a password (refer to guidelines outline in Section 5.5.1). |
| 2.4.6.8 | Shell and Python scripts cannot be ran without administrative privileges |
| 2.4.6.9 |  |
| 2.4.6.10 | Firewall protection is enabled. Refer to Section 5.5.1 under ‘Firewall Protection’ for additional details. |
| 2.4.6.11 | Application is completely separate from OS, with only minimal interaction through MQTT possible. |
| 2.4.7.1 | Firewall protection is enabled using firewall outlined in Section 3.5 |
| 2.4.7.4 | MQTT is currently the only support application protocol, using TCP. |
| 2.4.7.5 |  |
| 2.4.7.6 |  |
| 2.4.7.10 | WPA is disabled on authors router. |
| 2.4.7.13 |  |
| 2.4.7.18 |  |
|  |  |
| 2.4.8.4 | Ref. to 2.5.4 |
| 2.4.8.5 | Ref. to 2.5.4 |
| 2.4.8.6 | Ref. to 2.5.4 |
| 2.4.8.7 | Ref. to 3.3.1 |
| 2.4.8.8 | Ref. to 2.5.3 |
| 2.4.8.11 | No guest user accounts will be created |
| 2.4.8.12 |  |

Table 18: Proposed solutions for IoT Security best practices

# 8.0 Discussion

## 8.1 Results

The results of the experiment show confidently that it is possible to develop a secure surveillance system using a RPi device. In terms of algorithm selection, it can be safely said that the HOG classifier is the most suitable out of the algorithms attempted for a surveillance-based system, boasting an upwards of 88% accuracy. That, in conjunction with lower overall CPU stress and memory consumption, makes it a superior choice to the Haar-Cascades method when it comes to future systems. In terms of latency, the results clearly show that while implementing secure practices with the MQTT protocol does cause increases in latency, the differential is negligible and is definitely worth the small performance cost. In terms of portability, the system suffered when it came to this variable, meaning it definitely did not meet the claims the Xamarin framework is made out to be. Finally, this experiment proved it is more than possible to secure a device against common cyber-attacks against IoT devices. Using strategies such as hash *salting* passwords, enforcing secure password standards and using firewall protection proved an effective strategy against dictionary attacks.

## 8.2 Project Limitations

In terms of project limitations, access to any form of iPhone emulator was not possible without the use of a Mac, meaning it was not possible to test the application on this platform, defeating the purpose of using a hybrid application development framework. The app suffered from poor portability when attempted to be used on a mac device, resulting in multiple issues and highlighting its poor portability. The Xamarin framework is only recommended for those that have access to a Mac device as then it is truly as hybrid development framework, unlike its role in this thesis.

# 9.0 Conclusion

This thesis aimed to develop a secure Raspberry Pi surveillance camera with smart phone app integration. After determining the biggest cyber security threats against IoT devices and using the literature available to develop strategies against said attacks, the results clearly show a secure system. It can safely be assumed that following the IoT security guidelines outlined by the IoT Security Foundation can result in a secure system.

## 9.1 Future Work

The results of the thesis clearly showed it is possible to develop a Raspberry Pi-based surveillance system with adequate security. The limited number of face recognition algorithms used in the experiment cannot accurately conclude the best algorithm for the Raspberry Pi board, though can suggest Histogram of Gradients being a more than suitable candidate for similar projects. Due to time constraints on this thesis, the results of the accuracy of each algorithm cannot be truly determined, as the sample size of the data is far too small.

# 9.0 References

1. Merriman, C. (2019). *Google Cloud burst downs G Suite, Nest and Snapchat during 4-hour outage | The INQUIRER*. [online] http://www.theinquirer.net. Available at: https://www.theinquirer.net/inquirer/news/3076657/google-cloud-outage [Accessed 7 Jun. 2019].
2. Wang, A. (2018). ‘I’m in your baby’s room’: A hacker took over a baby monitor and broadcast threats, parents say. [online] Washington Post. Available at: https://www.washingtonpost.com/technology/2018/12/20/nest-cam-baby-monitor-hacked-kidnap-threat-came-device-parents-say/?noredirect=on&utm\_term=.04321c476652 [Accessed 7 Jun. 2019].
3. Ashford, W. (2019). *UK gears up for new laws on IoT security*. [online] ComputerWeekly.com. Available at: https://www.computerweekly.com/news/252462505/UK-gears-up-for-new-laws-on-IoT-security [Accessed 7 Jun. 2019].
4. Aziz, T. and Haq, E. (2018). Security Challenges Facing IoT Layers and its Protective Measures. *International Journal of Computer Applications*, [online] 179(27), pp.31-35. Available at: https://www.researchgate.net/publication/323892938\_Security\_Challenges\_Facing\_IoT\_Layers\_and\_its\_Protective\_Measures [Accessed 9 Jun. 2019].
5. Prabhakar, S. (2017). Network Security in Digitization: Attacks and Defence. *International Journal of Research in Computer Applications and Robotics,* [online] Vol. 5 Issue 5(ISSN 2320-7345). Available at: https://www.ijrcar.com/Volume\_5\_Issue\_5/v5i512.pdf [Accessed 9 Jun. 2019].
6. Conti, M., Dragoni, N. and Lesyk, V. (2016). A Survey of Man in The Middle Attacks. *IEEE Communications Surveys & Tutorials*, [online] 18(3), pp.2027-2051. Available at: https://ieeexplore.ieee.org/document/7442758 [Accessed 9 Jun. 2019].
7. Burhan, M., Rehman, R., Khan, B. and Kim, B. (2018). IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey. *Sensors*, [online] 18(9), p.2796. Available at: https://www.researchgate.net/publication/327272757\_IoT\_Elements\_Layered\_Architectures\_and\_Security\_Issues\_A\_Comprehensive\_Survey [Accessed 9 Jun. 2019].
8. Noble, F. (2016). Comparison of OpenCV's feature detectors and feature matchers. *2016 23rd International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*. [online] Available at: https://ieeexplore.ieee.org/document/7827292 [Accessed 26 May 2019].
9. Diachok, R., Dunets, R. and Klym, H. (2018). System of detection and scanning bar codes from Raspberry Pi web camera. *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*. [online] Available at: https://ieeexplore.ieee.org/document/8409124 [Accessed 26 May 2019].
10. Abdalla, G. and Veeramanikandasamy, T. (2017). Implementation of spy robot for a surveillance system using Internet protocol of Raspberry Pi. *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. [online] Available at: https://ieeexplore.ieee.org/document/8256563 [Accessed 26 May 2019].
11. Alam, M., Reaz, M. and Ali, M. (2012). A Review of Smart Homes—Past, Present, and Future. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, [online] 42(6), pp.1190-1203. Available at: https://ieeexplore.ieee.org/document/6177682 [Accessed 9 Jun. 2019].
12. Manoharan, R. and Chandrakala, S. (2015). Android OpenCV based effective driver fatigue and distraction monitoring system. *2015 International Conference on Computing and Communications Technologies (ICCCT)*. [online] Available at: https://ieeexplore.ieee.org/document/7292757 [Accessed 10 Jun. 2019].
13. Rastovich, R. (2015). *Why the MQTT Protocol is Ideal for the Internet of Things – ThingLogix, Inc*. [online] Thinglogix.com. Available at: https://www.thinglogix.com/why-the-mqtt-protocol-is-ideal/ [Accessed 10 Jun. 2019].
14. Liang, L., Zheng, K., Sheng, Q. and Huang, X. (2016). A Denial of Service Attack Method for an IoT System. *2016 8th International Conference on Information Technology in Medicine and Education (ITME)*. [online] Available at: https://ieeexplore.ieee.org/document/7976501 [Accessed 12 Jun. 2019].
15. Ssh.com. (2019). *SSH Protocol – Secure Remote Login and File Transfer | SSH.COM*. [online] Available at: https://www.ssh.com/ssh/protocol/#sec-How-to-Cite-SSH [Accessed 13 Jun. 2019].
16. Ylonen, T. (1996). SSH - Secure Login Connections over the Internet. *Proceedings of the 6th USENIX Security Symposium*, pp.37-42.
17. Abo-Soliman, M. and Azer, M. (2017). A study in WPA2 enterprise recent attacks. *2017 13th International Computer Engineering Conference (ICENCO)*. [online] Available at: https://ieeexplore.ieee.org/document/8289808 [Accessed 13 Jun. 2019].
18. Sreeraj, S. and Kumar, G. (2018). Performance of IoT protocols under constrained network, a Use Case based approach. *2018 International Conference on Communication, Computing and Internet of Things (IC3IoT)*. [online] Available at: https://ieeexplore.ieee.org/document/8668105 [Accessed 13 Jun. 2019].
19. Singh, M., Rajan, M., Shivraj, V. and Balamuralidhar, P. (2015). Secure MQTT for Internet of Things (IoT). *2015 Fifth International Conference on Communication Systems and Network Technologies*. [online] Available at: https://ieeexplore.ieee.org/abstract/document/7280018 [Accessed 13 Jun. 2019].
20. How safe are home security systems? (2015). [ebook] HP. Available at: http://go.saas.hpe.com/l/28912/2015-07-21/32bhy5/28912/69170/IoT\_Home\_Security\_Systems.pdf [Accessed 13 Jun. 2019].
21. Goshwe, N. (2013). Data Encryption and Decryption Using RSA Algorithmin a Network Environment. *IJCSNS International Journal of Computer Science and Network Security*, [online] VOL. 13(No. 7). Available at: http://paper.ijcsns.org/07\_book/201307/20130702.pdf [Accessed 16 Jun. 2019].
22. Kakar, V. (2016). Secure communication and authentication using Raspberry Pi. *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. [online] Available at: https://ieeexplore.ieee.org/document/7785011 [Accessed 16 Jun. 2019].
23. Patil, P., Narayankar, P., Narayan D.G. and Meena S.M. (2016). A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish. *Procedia Computer Science*, [online] 78, pp.617-624. Available at: https://www.sciencedirect.com/science/article/pii/S1877050916001101?via%3Dihub [Accessed 16 Jun. 2019].
24. Lee, S., Kim, H., Hong, D. and Ju, H. (2013). Correlation analysis of MQTT loss and delay according to QoS level. *The International Conference on Information Networking 2013 (ICOIN)*. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7428475 [Accessed 17 Jun. 2019].
25. Ludovici, A., Moreno, P. and Calveras, A. (2013). TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS. *Journal of Sensor and Actuator Networks*, [online] 2(2), pp.288-315. Available at: https://www.researchgate.net/publication/276039057\_TinyCoAP\_A\_Novel\_Constrained\_Application\_Protocol\_CoAP\_Implementation\_for\_Embedding\_RESTful\_Web\_Services\_in\_Wireless\_Sensor\_Networks\_Based\_on\_TinyOS [Accessed 17 Jun. 2019].
26. Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Systems Engineering Symposium (ISSE)*. [online] Available at: https://ieeexplore.ieee.org/document/8088251 [Accessed 17 Jun. 2019].
27. UK Government advice on Password Guidance, Simplifying your approach. (2015). [ebook] CESG and CPNI. Available at: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment\_data/file/458857/Password\_guidance\_-\_simplifying\_your\_approach.pdf [Accessed 18 Jun. 2019].
28. Fail2ban.org. (n.d.). *Fail2ban*. [online] Available at: https://www.fail2ban.org/wiki/index.php/Main\_Page [Accessed 18 Jun. 2019].
29. Cable. (2019). *Worldwide Broadband Speed League 2018 | Cable.co.uk*. [online] Available at: https://www.cable.co.uk/broadband/speed/worldwide-speed-league/ [Accessed 20 Jun. 2019].
30. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. and Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, [online] 17(4), pp.2347-2376. Available at: https://ieeexplore.ieee.org/document/7123563 [Accessed 20 Jun. 2019].
31. Krebs, B. (2017). *Beating JSON performance with Protobuf*. [online] Auth0 - Blog. Available at: https://auth0.com/blog/beating-json-performance-with-protobuf/ [Accessed 20 Jun. 2019].
32. Lin, B., Chen, Y., Chen, X. and Yu, Y. (2012). Comparison between JSON and XML in Applications Based on AJAX. *2012 International Conference on Computer Science and Service System*. [online] Available at: https://ieeexplore.ieee.org/document/6394535 [Accessed 20 Jun. 2019].
33. Sumaray, A. and Makki, S. (2012). A comparison of data serialization formats for optimal efficiency on a mobile platform. *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication - ICUIMC '12*. [online] Available at: https://dl.acm.org/citation.cfm?id=2184810 [Accessed 20 Jun. 2019].
34. IoT Security Compliance Framework. (2018). [ebook] Available at: https://www.iotsecurityfoundation.org/wp-content/uploads/2018/12/IoTSF-IoT-Security-Compliance-Framework-Release-2.0-December-2018.pdf [Accessed 17 Jun. 2019].
35. Raspbian.org. (n.d.). *FrontPage - Raspbian*. [online] Available at: https://www.raspbian.org/ [Accessed 21 Jun. 2019].
36. Raspberrypi.org. (n.d.). *Securing your Raspberry Pi - Raspberry Pi Documentation*. [online] Available at: https://www.raspberrypi.org/documentation/configuration/security.md [Accessed 21 Jun. 2019].
37. George, G. (2019). *How to secure your Raspberry Pi board [Tutorial] | Packt Hub*. [online] Packt Hub. Available at: https://hub.packtpub.com/how-to-secure-your-raspberry-pi-board-tutorial/ [Accessed 21 Jun. 2019].
38. Uk.norton.com. (n.d.). *What Is A Botnet?* [online] Available at: https://uk.norton.com/internetsecurity-malware-what-is-a-botnet.html [Accessed 21 Jun. 2019].
39. Krebs, B. (2017). *Reaper: Calm Before the IoT Security Storm? — Krebs on Security*. [online] Krebsonsecurity.com. Available at: https://krebsonsecurity.com/2017/10/reaper-calm-before-the-iot-security-storm/ [Accessed 21 Jun. 2019].
40. Dlib.net. (n.d.). *dlib C++ Library*. [online] Available at: http://dlib.net/ [Accessed 24 Jun. 2019].
41. Docs.opencv.org. (n.d.). *OpenCV: Face Detection using Haar Cascades*. [online] Available at: https://docs.opencv.org/3.4.1/d7/d8b/tutorial\_py\_face\_detection.html [Accessed 24 Jun. 2019].
42. Treacy, F. (2019). *HART Communication Made Easy*. [ebook] Available at: https://www.analog.com/media/en/technical-documentation/technical-articles/HART\_Communication\_Made\_Easy.pdf [Accessed 24 Jun. 2019].
43. Instrumentation and Control Engineering. (2018). *HART (Highway Addressable Remote Transducer) Communication Protocol*. [online] Available at: https://automationforum.co/hart-highway-addressable-remote-transducer-communication-protocol/ [Accessed 24 Jun. 2019].
44. Redis.io. (n.d.). *Redis Protocol specification – Redis*. [online] Available at: https://redis.io/topics/protocol [Accessed 25 Jun. 2019].
45. Patel, K. and Upadhyay, H. (2018). *A Rule based Approach to Mitigate DDoS attack in IoT Environment*. [ebook] Available at: https://pdfs.semanticscholar.org/a8af/1a108244b37d1994962e5f250875f0adb1ab.pdf [Accessed 25 Jun. 2019].
46. Abad, C. and Bonilla, R. (2007). An Analysis on the Schemes for Detecting and Preventing ARP Cache Poisoning Attacks. *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*. [online] Available at: https://ieeexplore.ieee.org/document/4279062 [Accessed 28 Jun. 2019].
47. GitHub. (2019). *jrosebr1/imutils*. [online] Available at: https://github.com/jrosebr1/imutils [Accessed 28 Jun. 2019].
48. GeeksforGeeks. (n.d.). *Understanding Python Pickling with example - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/understanding-python-pickling-example/ [Accessed 28 Jun. 2019].
49. GitHub. (2019). *ageitgey/face\_recognition*. [online] Available at: https://github.com/ageitgey/face\_recognition [Accessed 28 Jun. 2019].
50. Pawar, A. and Umale, V. (2018). Internet of Things Based Home Security Using Raspberry Pi. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8697345 [Accessed 5 Jul. 2019].
51. Hussein, N. and Al Mansoori, I. (2017). Smart Door System for Home Security Using Raspberry pi3. *2017 International Conference on Computer and Applications (ICCA)*. [online] Available at: https://ieeexplore.ieee.org/document/8079785 [Accessed 5 Jul. 2019].
52. Tatli, E. (2015). Cracking More Password Hashes with Patterns. *IEEE Transactions on Information Forensics and Security*, [online] 10(8), pp.1656-1665. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7084670 [Accessed 18 Jul. 2019].
53. Arias, D. (2018). *Adding Salt to Hashing: A Better Way to Store Passwords*. [online] Auth0 - Blog. Available at: https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/ [Accessed 18 Jul. 2019].
54. Patel, H. and Rajput, R. (2018). Smart Surveillance System Using Histogram of Oriented Gradients (HOG) Algorithm and Haar Cascade Algorithm. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. [online] Available at: https://ieeexplore.ieee.org/document/8697464 [Accessed 19 Jul. 2019].
55. Noman, M., Yousaf, M. and Velastin, S. (2016). An Optimized and Fast Scheme for Real-Time Human Detection Using Raspberry Pi. *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. [online] Available at: https://ieeexplore.ieee.org/document/7797008 [Accessed 19 Jul. 2019].
56. Sqlite.org. (n.d.). *Distinctive Features of SQLite*. [online] Available at: https://www.sqlite.org/different.html [Accessed 19 Jul. 2019].
57. GitHub. (2019). *eclipse/paho.mqtt.m2mqtt*. [online] Available at: https://github.com/eclipse/paho.mqtt.m2mqtt [Accessed 19 Jul. 2019].
58. Eclipse Mosquitto. (n.d.). *Eclipse Mosquitto*. [online] Available at: https://mosquitto.org/ [Accessed 27 Jul. 2019].
59. Letsencrypt.org. (n.d.). *Getting Started - Let's Encrypt - Free SSL/TLS Certificates*. [online] Available at: https://letsencrypt.org/getting-started/ [Accessed 27 Jul. 2019].
60. Hivemq.com. (2015). *MQTT Message Data Integrity - MQTT Security Fundamentals*. [online] Available at: https://www.hivemq.com/blog/mqtt-security-fundamentals-mqtt-message-data-integrity/ [Accessed 28 Jul. 2019].
61. Blumtritt, C. (2019). *Revenue in the Smart Home segment Security worldwide\* 2023 | Statista*. [online] Statista. Available at: https://www.statista.com/statistics/645392/revenue-in-the-smart-home-segment-security-worldwide/ [Accessed 12 Jun. 2019].
62. Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F. and Alonso-Zarate, J. (2015). A Survey on Application Layer Protocols for the Internet of Things. *CSSN Research Lab, Department of Informatics, Alexander TEI of Thessaloniki, Greece*. [online] Available at: https://www.researchgate.net/profile/Periklis\_Chatzimisios/publication/303192188\_A\_survey\_on\_application\_layer\_protocols\_for\_the\_Internet\_of\_Things/links/577b656608ae213761c9d91d.pdf [Accessed 13 Aug. 2019].
63. Albazrqaoe, W., Huang, J. and Xing, G. (2019). A Practical Bluetooth Traffic Sniffing System: Design, Implementation, and Countermeasure. *IEEE/ACM Transactions on Networking*, [online] 27(1), pp.71-84. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8556096 [Accessed 13 Aug. 2019].
64. Kvarda, L., Hnyk, P., Vojtech, L., Lokaj, Z., Neruda, M. and Zitta, T. (2016). Software Implementation of a Secure Firmware Update Solution in an IOT Context. *Advances in Electrical and Electronic Engineering*, [online] 14(4). Available at: https://pdfs.semanticscholar.org/7515/43344eee3b7716e4fbcdcf57c6520de0afe0.pdf [Accessed 13 Aug. 2019].
65. Elizalde, D. (2019). *How to Protect Your IoT Product from Hackers*. [online] Daniel Elizalde. Available at: https://danielelizalde.com/iot-security-hacks-worst-case-scenario/ [Accessed 13 Aug. 2019].
66. Putri Ratna, A., Dewi Purnamasari, P., Shaugi, A. and Salman, M. (2013). Analysis and comparison of MD5 and SHA-1 algorithm implementation in Simple-O authentication based security system. *2013 International Conference on QiR*. [online] Available at: https://www.researchgate.net/publication/261304772\_Analysis\_and\_comparison\_of\_MD5\_and\_SHA-1\_algorithm\_implementation\_in\_Simple-O\_authentication\_based\_security\_system [Accessed 13 Aug. 2019].
67. Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. [online] Available at: https://ieeexplore.ieee.org/document/990517 [Accessed 1 Aug. 2019].
68. Sqlitebrowser.org. (2019). *DB Browser for SQLite*. [online] Available at: https://sqlitebrowser.org/ [Accessed 16 Aug. 2019].
69. SwitchDoc Labs. (2014). *Reliable Projects 1: WatchDog Timers for Raspberry Pi and Arduinos - SwitchDoc Labs*. [online] Available at: https://www.switchdoc.com/2014/11/reliable-projects-watchdog-timers-raspberry-pi-arduinos/ [Accessed 17 Aug. 2019].
70. Docs.microsoft.com. (2019). *System.Security.Cryptography Namespace*. [online] Available at: https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography?view=netframework-4.8 [Accessed 18 Aug. 2019].
71. Bamburic, M. (2019). *The best new features in Visual Studio 2019 (and Visual Studio 2019 for Mac)*. [online] BetaNews. Available at: https://betanews.com/2019/04/02/best-new-features-visual-studio-2019/ [Accessed 20 Aug. 2019].
72. Harmoush, E. (2015). *Message Integrity – Practical Networking .net*. [online] Practical Networking .net. Available at: https://www.practicalnetworking.net/series/cryptography/message-integrity/ [Accessed 20 Aug. 2019].
73. Mehmood, A. (2017). *Differences between Hash functions, Symmetric & Asymmetric Algorithms*. [online] Cryptomathic.com. Available at: https://www.cryptomathic.com/news-events/blog/differences-between-hash-functions-symmetric-asymmetric-algorithms [Accessed 20 Aug. 2019].
74. Villanueva, J. (2015). *What is a Digital Signature?*. [online] Jscape.com. Available at: https://www.jscape.com/blog/what-is-a-digital-signature [Accessed 20 Aug. 2019].
75. Suchitra, Suja P. and Tripathi, S. (2016). Real-time emotion recognition from facial images using Raspberry Pi II. *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*. [online] Available at: https://ieeexplore.ieee.org/document/7566780 [Accessed 20 Aug. 2019].
76. Dlib.net. (2019). *dlib C++ Library*. [online] Available at: http://dlib.net/ [Accessed 20 Aug. 2019].
77. Cherecwich, P. (2019). *Security vs. Convenience: The Ultimate Trade-Off?*. [online] Medium. Available at: https://medium.com/face-value/security-vs-convenience-the-ultimate-trade-off-843e476bf538 [Accessed 20 Aug. 2019].

# Appendices

## Appendix A – Raspberry Pi Specifications

|  |  |
| --- | --- |
| **Specification** |  |
| SoC | Broadcom BCM2837 |
| CPU | 4× ARM Cortex-A53, 1.2GHz |
| GPU | Broadcom VideoCore IV |
| **RAM** | 1GB LPDDR2 (900 MHz) |
| **Networking** | 10/100 Ethernet, 2.4GHz 802.11n wireless |
| **Bluetooth** | Bluetooth 4.1 Classic, Bluetooth Low Energy |
| **Storage** | microSD |
| **GPIO** | 40-pin header, populated |
| **Ports** | HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI) |

1. https://github.com/ChrisPaulDillon/MSc\_SurveillanceCam [↑](#footnote-ref-1)
2. https://github.com/ChrisPaulDillon/MSc\_SmartHomeApp [↑](#footnote-ref-2)
3. https://github.com/ChrisPaulDillon/MSc\_SurveillanceCam/tree/master/DataAnalysis [↑](#footnote-ref-3)