

**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

TODO

# Acknowledgments

TO DO

Table of Contents

[Table of Figures 5](#_Toc15564752)

[Table of Tables 6](#_Toc15564753)

[1.0 Introduction 7](#_Toc15564754)

[1.1 Background 7](#_Toc15564755)

[1.2 Problem Description 8](#_Toc15564756)

[1.3 Project objectives 9](#_Toc15564757)

[2.0 Literature Review 10](#_Toc15564758)

[2.1 Internet of Things Architecture 10](#_Toc15564759)

[2.1.1 Perception Layer 10](#_Toc15564760)

[2.1.2 Network Layer 11](#_Toc15564761)

[2.1.3 Application Layer 11](#_Toc15564762)

[2.2 Network Communication Protocols 11](#_Toc15564763)

[2.2.1 Bluetooth 12](#_Toc15564764)

[2.2.2 Wi-Fi 12](#_Toc15564765)

[2.2.3 ZigBee 12](#_Toc15564766)

[2.2.4 HART 12](#_Toc15564767)

[2.3 Application Communication Protocols 13](#_Toc15564768)

[2.3.1 Message Queue Telemetry Transport 13](#_Toc15564769)

[2.3.2 Advanced Message Queueing Protocol 14](#_Toc15564770)

[2.3.3 Constrained Application Protocol 15](#_Toc15564771)

[2.3.4 REdis Serialization Protocol 15](#_Toc15564772)

[2.4 Data Exchange Formats 17](#_Toc15564773)

[2.4.1 XML 17](#_Toc15564774)

[2.4.2 JSON 17](#_Toc15564775)

[2.4.3 Protocol Buffers 18](#_Toc15564776)

[2.5 Security Threats 19](#_Toc15564777)

[2.5.1 DOS Attack 19](#_Toc15564778)

[2.5.2 Man in The Middle Attack 20](#_Toc15564779)

[2.5.3 Brute Force Attacks 21](#_Toc15564780)

[2.5.4 Malicious Code Attack 21](#_Toc15564781)

[2.5.5 Physical Tampering 21](#_Toc15564782)

[2.6 Cryptography Options 22](#_Toc15564783)

[2.6.1 Symmetric Encryption 22](#_Toc15564784)

[2.6.2 Asymmetric Encryption 22](#_Toc15564785)

[2.6.3 Hash Encryption 23](#_Toc15564786)

[2.6.4 Salt Encryption 24](#_Toc15564787)

[2.7 Message Integrity 24](#_Toc15564788)

[2.7.1 Hash Algorithms 24](#_Toc15564789)

[2.7.2 Message Authentication Code Algorithms 24](#_Toc15564790)

[2.7.3 Digital Signatures 24](#_Toc15564791)

[2.8 Facial Recognition 25](#_Toc15564792)

[2.8.1 Haar Cascade 25](#_Toc15564793)

[2.8.2 Histogram of Oriented Gradients 26](#_Toc15564794)

[2.9 Security Guidelines for Internet of Things 27](#_Toc15564795)

[3.0 Technology Review 28](#_Toc15564796)

[3.1 Hardware Selection 28](#_Toc15564797)

[3.2 Operating System Selection 28](#_Toc15564798)

[3.2.1 Raspberry Pi Operating System 28](#_Toc15564799)

[3.2.2 Cyber Security Operating System 28](#_Toc15564800)

[3.3 Programming Language Selection 29](#_Toc15564801)

[3.4 Library Selection 29](#_Toc15564802)

[3.4.1 Python Library Selection 29](#_Toc15564803)

[3.4.2 C# Library Selection 30](#_Toc15564804)

[3.5 Software Review 31](#_Toc15564805)

[4.0 System Requirements 32](#_Toc15564806)

[4.1 Hardware and Software Requirements 32](#_Toc15564807)

[4.2 Functional & Non Functional Requirements 33](#_Toc15564808)

[4.3 Raspberry Pi Specifications 34](#_Toc15564809)

[5.0 Methodology 35](#_Toc15564810)

[5.1 Software Development Methodology 35](#_Toc15564811)

[5.2 Main Flow Design & Overview 36](#_Toc15564812)

[5.3 Surveillance System Development 37](#_Toc15564813)

[5.3.1 Machine-To-Machine Communication 37](#_Toc15564814)

[5.3.2 Gathering Face Dataset 38](#_Toc15564815)

[5.3.3 Encoding Face Dataset 39](#_Toc15564816)

[5.3.4 Real Time Face Recognition 39](#_Toc15564817)

[5.3.5 MQTT Communication 41](#_Toc15564818)

[5.3.6 Power Outage Considerations 43](#_Toc15564819)

[5.4 Application Development 43](#_Toc15564820)

[5.4.1 Initial Class Diagram Model 43](#_Toc15564821)

[5.4.2 MQTT Communication 44](#_Toc15564822)

[5.4.3 Account Creation and Login 45](#_Toc15564823)

[5.4.3 Database Development 46](#_Toc15564824)

[5.4.4 Displaying Received Images 46](#_Toc15564825)

[5.4.5 Password Storage 46](#_Toc15564826)

[5.5 Security Implementation 47](#_Toc15564827)

[5.5.1 Raspberry Pi Specific Security 47](#_Toc15564828)

[5.5.2 Firewall Protection 47](#_Toc15564829)

[5.5.3 MQTT Communication Security 48](#_Toc15564830)

[5.6 Security Best Practices 49](#_Toc15564831)

[7.0 References 50](#_Toc15564832)

# Table of Figures

[Figure 1: Smart Home Security Revenue forecast worldwide (Blumtritt, 2019) 9](#_Toc16523019)

[Figure 2: IoT three layered architecture in relation to a home surveillance system 13](#_Toc16523020)

[Figure 3: MQTT Architecture using a Raspberry Pi as both a Publisher and Broker 16](#_Toc16523021)

[Figure 4: Basic AMQP protocol process with exchange and queue system 17](#_Toc16523022)

[Figure 5: Basic GET/Response between a CoAP Client & Server 17](#_Toc16523023)

[Figure 6: Code Snippet showing basic XML Schema 19](#_Toc16523024)

[Figure 7: Code snippet showing basic JSON Schema 19](#_Toc16523025)

[Figure 8: A DDoS attack being carried out using infected PCs (slaves) 21](#_Toc16523026)

[Figure 9: MITM attack showing an intercepted connection from a malicious attacker 22](#_Toc16523027)

[Figure 10: Computer running a malicious script to attempt logins on a website 23](#_Toc16523028)

[Figure 11: Example of plain text being encrypted using symmetric key encryption 24](#_Toc16523029)

[Figure 12: Example of plain text being encrypted using Asymmetric key encryption 25](#_Toc16523030)

[Figure 13: SHA-1 & MD5 brute force successful attack times (Putri Ratna et al., 2013) 25](#_Toc16523031)

[Figure 14: Most commonly detect features in the Haar Cascade algorithm 27](#_Toc16523032)

[Figure 15: Raspberry Pi Model 3b 30](#_Toc16523033)

[Figure 16: Incremental Software Development Lifecycle for a Home Surveillance System 37](#_Toc16523034)

[Figure 17: Main Interaction of the Surveillance System and Smart Phone Application 38](#_Toc16523035)

[Figure 18: SSH Protocol for Machine-To-Machine Communication (ssh.com, 2019) 39](#_Toc16523036)

[Figure 19: Example of using face\_recognition support library to receive faces 41](#_Toc16523037)

[Figure 20: Mosquitto broker (RPi) interacting with Paho & M2Mqtt clients on respective devices 44](#_Toc16523038)

[Figure 21: UML Class Diagram highlighting the relationships between the user and MQTT interaction 46](#_Toc16523039)

[Figure 22: Initial User Interface Design for Viewing Images within the App 46](#_Toc16523040)

[Figure 23: Initial UI Design for Login and Registration 48](#_Toc16523041)

[Figure 24: MQTT Communication secured using Certification and TLS 50](#_Toc16523042)

[Figure 25: Difference between a false negative and true positive result in facial recognition 51](#_Toc16523043)

[Figure 26: Difference in Computational Stress between the HOG and Haar algorithms 52](#_Toc16523044)

[Figure 27: Difference in Memory Consumption between the HOG and Haar algorithms using 10 encoded images 53](#_Toc16523045)

# Table of Tables

[Table 1: Comparison of common IoT Network Communication Protocols 15](#_Toc16522098)

[Table 2: Features of most common IoT Application Communication Protocols 18](#_Toc16522099)

[Table 3: Serialized size in bytes (Sumaray and Makki, 2012) 20](#_Toc16522100)

[Table 4: Average serialization time in milliseconds (Sumaray and Makki, 2012) 20](#_Toc16522101)

[Table 5: Average deserialization time in milliseconds (Sumaray and Makki, 2012) 20](#_Toc16522102)

[Table 6: Relevant IoT best practices in relation to the project 29](#_Toc16522103)

[Table 7: Hardware Requirements for project 34](#_Toc16522104)

[Table 8: Software Requirements for project 34](#_Toc16522105)

[Table 9: Core Functional Requirements 35](#_Toc16522106)

[Table 10: Non-Functional Requirements 35](#_Toc16522107)

[Table 11: Raspberry Pi Model 3b Specifications 36](#_Toc16522108)

[Table 12: HOG Algorithm Accuracy recorded using different volumes of encoded images 52](#_Toc16522109)

[Table 13: Haar-Cascades Algorithm Accuracy using different volumes of encoded images 52](#_Toc16522110)

[Table 14: Comparison of CPU & Memory usage between the HOG & Haar algorithms 53](#_Toc16522111)

[Table 15: Proposed solutions for IoT Security best practices 55](#_Toc16522112)

# 1.0 Introduction

## 1.1 Background

With the recent explosion of Internet of Things devices, 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 CCTV; 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. Building a custom made home surveillance system poses various security challenges similar to all IoT devices. Strategies such as local base storage, implementation of cryptography-based MQTT communication and salt hashing algorithms are proposed to deal with the various challenges present at both the application and network layer. By investigating the best security practices and covering every layer of the IoT architecture, 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 a Pi 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:

* Authorised access to the RPi (authentication)
* Integrated authentication for the smartphone app and IoT platform
* Explore security through restricting port access
* Storage security through encryption and reducing exposure of data to the network

These will be created in accordance with the best practices outlined by the IoT security foundation (IoT Security Compliance Framework, 2018). If time permits, a separate machine can be set up with the Kali Linux operating system, an operating system used in cybersecurity and digital forensics. The machine can perform various cyber-attacks (sniffing, MITM, DoS) and evaluate how the system performs against them. A recent paper demonstrates various types of DoS attacks were carried out in a testbed environment against an IoT device, resulting in a successful attack against the network it was hosted on (Liang et al., 2016).

## 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.
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.
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.
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.
7. Develop the device
8. Test the device

# 2.0 Literature Review

RPi-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 Pi 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. Finally, the system 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 2: 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 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
* Commonplace in-home, to increase scalability
* High levels of security capability (encryption & authentication)
* Capable of fast data transmission (50mb/s+)

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.

### 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).

### 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 provides multiple security vulnerabilities. The main issue being, by default, there is no encryption mechanism. This leaves the network prone to MITM attacks and DoS 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

This section reviews some of the most commonly used application communication protocols. These are 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. QoS Level 0 sends a message once and does not check if the message arrived at its given destination. 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. Finally, 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 3 also shows multiple smart phone clients subscribed to a specific topic awaiting a response.



Figure 3: 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 4). 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.

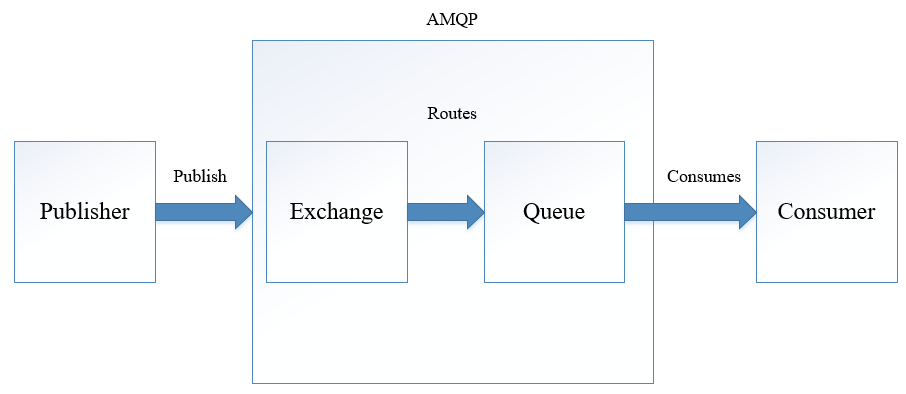


Figure 4: Basic AMQP protocol process with exchange and queue system

### 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 5). Non-confirmable messages have no such system in place.

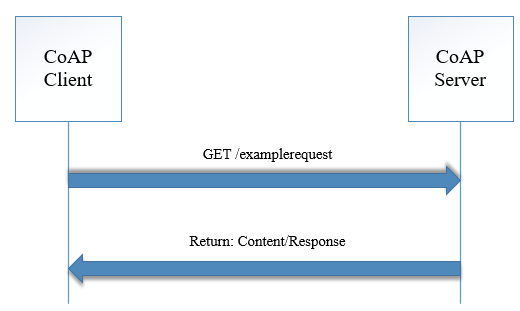


Figure 5: Basic GET/Response between a CoAP Client & Server

### 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
* Serialization Time
* Deserialization Time

### 2.4.1 XML

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. It is text-based, which can be problematic in terms of speed, as it will also need to be parsed character by character.



Figure 6: Code Snippet showing basic XML Schema

### 2.4.2 JSON

JSON or JavaScript Object Notation 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 7). 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 7: 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. 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.

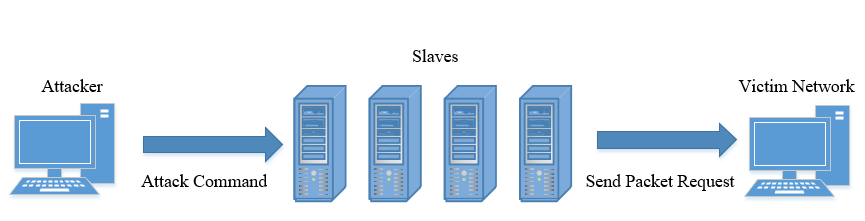


Figure 8: A DDoS attack being carried out using infected PCs (slaves)

### 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.

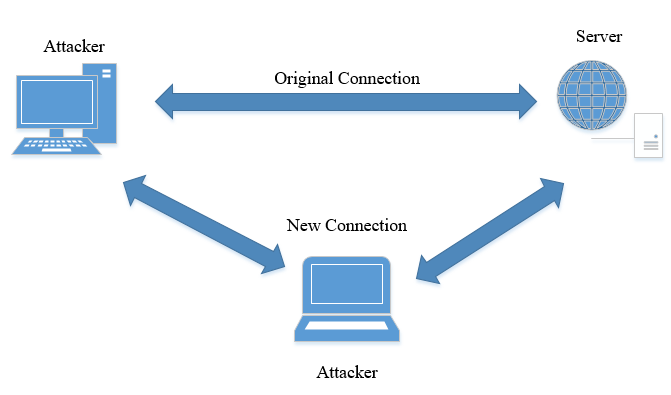


Figure 9: MITM attack showing an intercepted connection from a malicious attacker

### 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. 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.

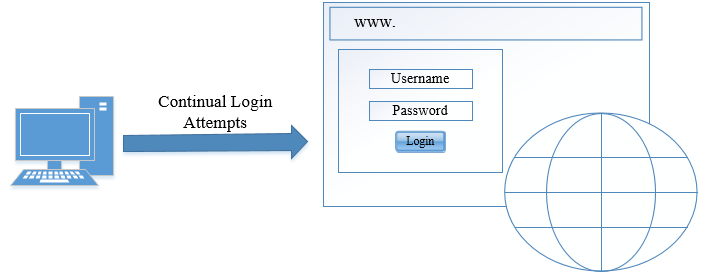


Figure 10: Computer running a malicious script to attempt logins on a website

### 2.5.4 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 cannot be detected by security tools such as anti-viruses as it not detected as a virus or malware.

### 2.5.5 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. 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.

## 2.6 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 is a symmetric form of encryption (Figure 11), 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).

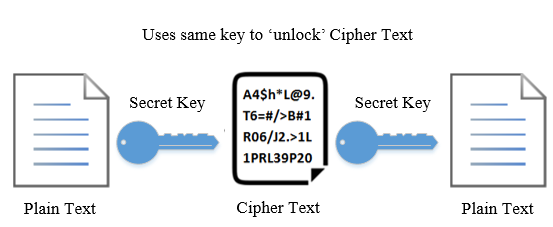


Figure 11: Example of plain text being encrypted using symmetric key encryption

### 2.6.2 Asymmetric Encryption

RSA works on a public-private key basis that is a form of asymmetric encryption (Figure 12). 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.

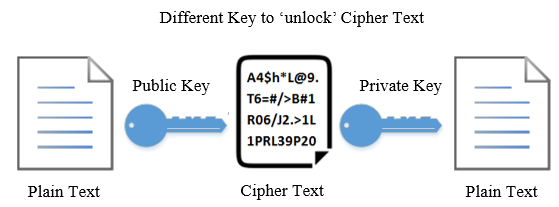


Figure 12: Example of plain text being encrypted using Asymmetric key encryption

### 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 11.



Figure 13: 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. 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 previously mentioned, hashing is an excellent way to achieve data integrity when transferring files as well as storing passwords securely. 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 (change?) and only to the payload, it has very little impact on message transport speed.

### 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 key encryption (see 2.5), meaning only trusted parties have access to the key required to create the digital stamp. This is a form of symmetric encryption.

### 2.7.3 Digital Signatures

Digital signatures improve upon the MAC method of securing MQTT messages by ensuring non-repudiations of the message sent. This utilises asymmetric key encryption, meaning only the sender of the message with access to the private key can generate the digital stamp. 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.

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 12). 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).

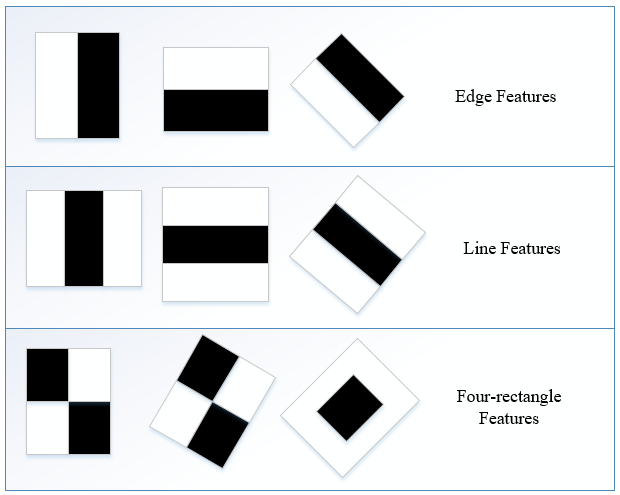


Figure 14: Most commonly detect features in the Haar Cascade algorithm

To remedy this, Adaboost is normally implemented. 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)

With the computational limitations of the Raspberry Pi considered, the HOG algorithm appears to be viable in terms of real time face detection. In their paper (Noman, Yousaf and Velastin, 2016), it is demonstrated that human detection in real time is possible using a simple RPi model. 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 Pi. 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 Pi, as the one proposed in this thesis.

## 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 systems, 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 Pi in a recent paper, demonstrating the boards capabilities. (Abdalla and Veeramanikandasamy, 2017).



Figure 15: Raspberry Pi Model 3b

## 3.2 Operating System Selection

### 3.2.1 Raspberry Pi Operating System

While there are multiple OS systems available for the Raspberry Pi, Raspbian is currently the recommend, having been designed specifically for the Raspberry Pi (Raspbian.org, 2019). 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

Currently, the standard operating system used in cyber security and digital forensics is Kali Linux, derived from the Linux distribution Debian. The 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

*Raspberry Pi Development*

Development on the Raspberry Pi has limitations in terms of programming language choice, with platform specific languages such as C# being severely limited due to having little support for non-Windows platforms. Python is a programming language which offers tremendous support in terms of Computer Vision, granting access to libraries OpenCV and Dlib. 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 Pi. 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 Pi 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.

### 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

## 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.

**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.

# 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 6) are established, with each hardware requirement having an additional backup in case of malfunction. Software requirements are also listed (Table 7) 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 8 lists the core functionality of the system with Table 9 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

## 4.3 Raspberry Pi Specifications

The following table has been constructed to detail the specifications for the Raspberry Pi used in this experiment:

|  |  |
| --- | --- |
| 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) |

Table 11: Raspberry Pi Model 3b Specifications

# 5.0 Methodology

The methodology section details the strategies in developing both the application and surveillance system while using security techniques outlined in Section 2.6. This includes the Software Development Lifecycle used to create the project, the Python scripts used for the Surveillance system to function and the application development which is used to interact with the Raspberry Pi. MQTT is used to bridge the communication between the devices. Finally, the system is secured using state of the art Cryptography techniques.

## 5.1 Software Development Methodology

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’. This project naturally falls into three parts, being Pi (surveillance system) requirements, application development requirements and security-specific non-functional requirements. 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 image-based:

1. Surveillance system development. This includes everything relating to the Pi such as face recognition, model training, capturing and encoding images, database storage, etc.
2. Application related 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 the implementation of the application communication protocol MQTT.
3. Security-related development. After the foundational functional requirements are met, security is incorporated into all three layers of the IoT architecture (refer to Section 2.1), with the requirements with the highest priority being accomplished first.

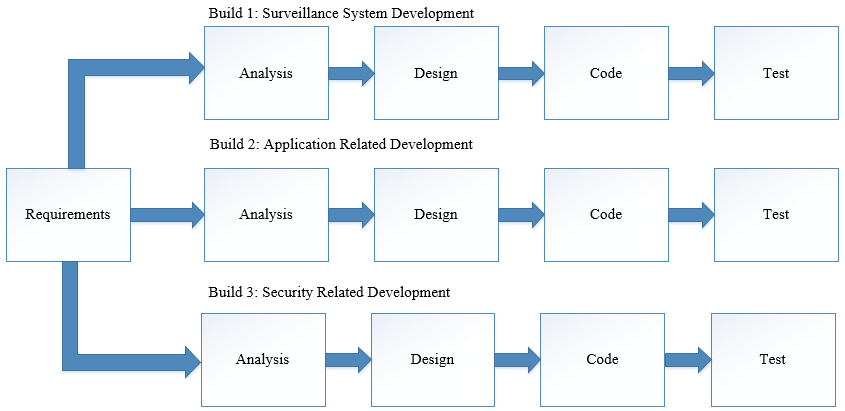


Figure 16: Incremental Software Development Lifecycle for a Home Surveillance System

## 5.2 Main Flow Design & Overview

The main flow of the system would initially begin after an unidentified individual enters the premises, prompting the Raspberry Pi to send the captured image through MQTT (shown in Figure 22). 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. This will only be available if the user is logged in, if not they will be prompted to login and can proceed from there. 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.

### 5.3.1 Machine-To-Machine Communication

Often when it comes to small board computers, they are difficult to access physically. In terms of the proposed system, the small device used as a surveillance system will be difficult to access without remote capabilities. A solution to this is the Secure Shell protocol or SSH as its commonly referred to. It is a method to remotely access one computing from another securely (Ylonen, 1996). Based on the client-server model, a connection is established by the SSH client by connecting to the SSH server (see Figure 18). The client then uses public key cryptography to verify the identity of the SSH server. For data exchange, the protocol uses symmetric encryption and hashing algorithms, ensuring the privacy and integrity of the data sent between the two devices. Although SSH will be utilised very frequently during development, it is not necessary to have after development, as the whole system is automated. Therefore, it is recommended to disable SSH and close its associated port to help in the preventing of unauthorised access.

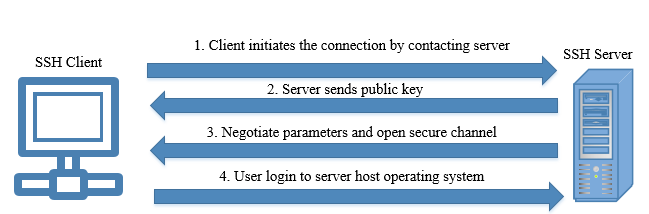


Figure 18: SSH Protocol for Machine-To-Machine Communication (ssh.com, 2019)

### 5.3.2 Gathering 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 Pseudocode showing while loop used to keep script running to capture images of an individual

### 5.3.3 Encoding Face Dataset

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 19).

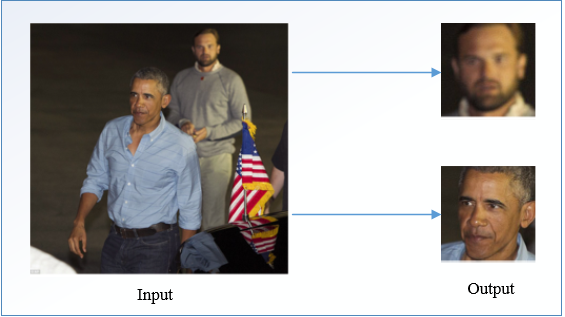


Figure 19: Example of using face\_recognition support library to receive faces

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 Pseudocode For loop cycling through all images in the 'dataset' folder

### 5.3.4 Real-Time Face Recognition

As discussed previously, Deep Learning approaches are very computationally expensive and function best running on graphics card instead of a central processing unit (CPU), therefore the HOG method has been selected.

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 Pseudocode showing frame being modified to speed 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 Pseudocode showing live facial comparison attempting face recognition 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 Pseudocode showing live labelling of identified/unidentified face +image being captured

### 5.3.5 MQTT Communication

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 has been selected as the broker, as it can be hosted on the Pi. The broker (server) will interact with the clients to send out messages on various topics (seen in Figure 19). 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#).

Both these clients support TLS communication which is essential in securing communication between the devices. The following settings are utilised by the Paho Client on the Pi:

*Client*: “Paho”

*Broker*: “raspberrypi”

*Client ID*: “smartphone”

*TLS*: “True”

*Port*: 8883

*QoS*: 2

*Publish Topic*: “cpd1995/surveillance”

*Subscription Topic*: “cpd1995/doorlock”

The broker name corresponds to the server certificate, being ‘raspberrypi’ in this case. The Client ID corresponds to the common name variable set when creating the Certificate. As detailed in Section 5.4.3, the communication between the devices is secured using TLS and certificate-based certification. Quality of Service 2 was selected as it was deemed most suitable for this project (see 2.3.1).

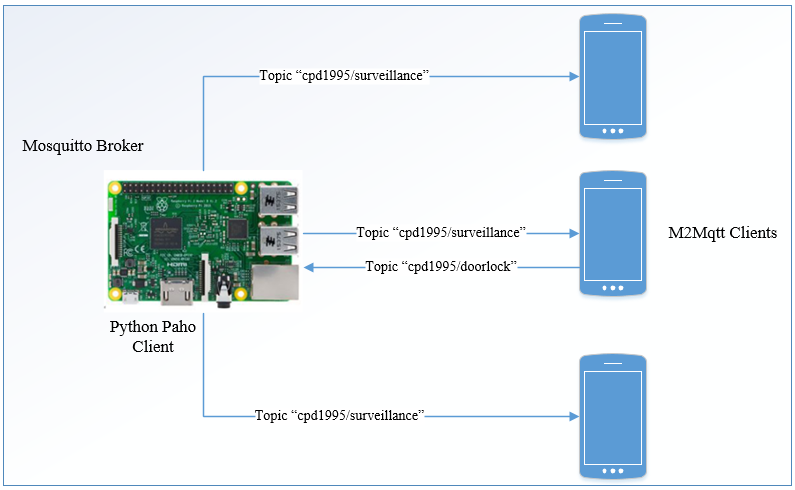


Figure 20: Mosquitto broker (RPi) interacting with Paho & M2Mqtt clients

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. This is done utilising the Mosquitto broker under port 8883 using certificate-based authentication (see 5.4.3) to ensure the data is sent securely.

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 6: Python Pseudocode showing publish method for Paho Client

### 5.3.6 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.

## 5.4 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.

### 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 2-factor authentication, the user must be logged in before being able to access images received via MQTT. Figure 20 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 21: UML Class Diagram highlighting the relationships between the user and MQTT interaction

### 5.4.2 MQTT Communication

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. The option to delete the image is also available on this screen (see Figure 21 for a prototype diagram).



Figure 22: Initial User Interface Design for Viewing Images within the App

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.

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 7: C# Pseudocode on handling messages received from MQTT

M2Mqtt was selected as a client for smartphone devices due to its compatibility with Xamarin and TLS support for Certificate-based communication. The following settings are utilised by this client:

*Client*: “M2Mqtt”

*Broker*: “raspberrypi”

*Client ID*: “smartphone”

*TLS*: “True”

*Port*: 8883

*QoS*: 2

*Subscription Topic*: “cpd1995/surveillance”

*Publish Topic*: “cpd1995/doorlock”

### 5.4.3 Account Creation and Login

Initially, the account creation page will be available for a family member or any authorised individuals to register an account (shown in Figure 23). This will be used to access the images and once the user logs in, every available image will be loaded from the database.

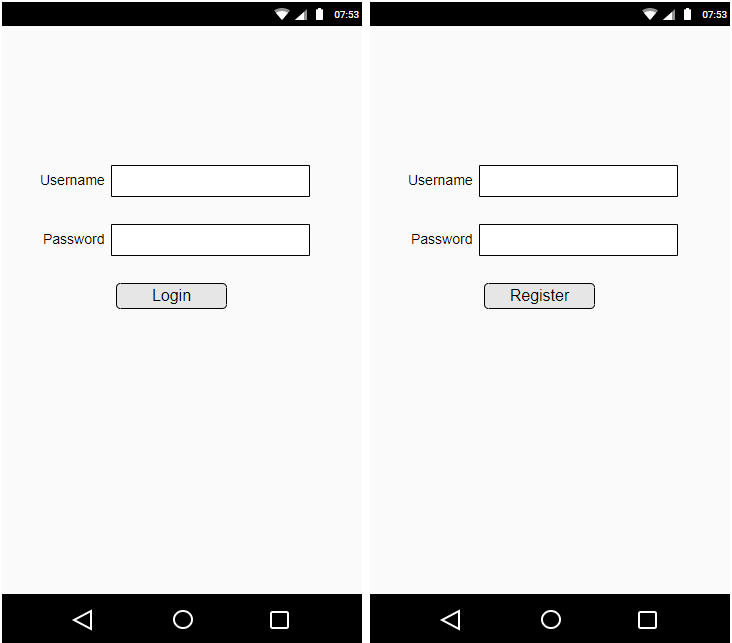


Figure 23: Initial UI Design for Login and Registration

### 5.4.3 Database Development

For the images to be stored and viewed by the user, SQLite has been proposed to store the image data along with the timestamp of when it was received. The image, encoded in Base64, can be stored in that format and reconstructed when requested by the user. Additionally, to grant access to the app, the user must log in, therefore an additional database must be created to store the usernames and password of each family member.

### 5.4.4 Displaying 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.

### 5.4.5 Password Storage

To access the images and application, a username and password must be supplied. This is the initial layer of authentication and security to verify the individual is who they say they are. As outlined in Section 2.6, password hashing along with a salt will be used to ensure that the passwords are safe in the case of a dictionary or brute force attempt by a malicious individual.

## 5.5 Security Implementation

The final step of the incremental lifecycle is securing the project using the various methods discussed in Section 2.6 and Section 2.7. With the core functionality finished, each individual component and layer of the project must be secured in accordance with the standards outlined by the IoT Security Foundation.

### 5.5.1 Raspberry Pi Specific Security

This section details security measures specific to the Pi board along with its predominately-used operating system, Raspbian. According to the Raspberry Pi official documentation, the following are considered best practices (Raspberrypi.org, 2019) for securing the device:

1. 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.
2. 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.
3. Ensure the latest security fixes. This means having the most recent up-to-date version of the operating system and ensuring SSH is up to date.
4. Use key-based authentication. As discussed in the Cryptography section there are various ways of encrypting a service. By forcing an encryption-based authentication, an additional layer of security is added as now there are two factors required to log in.
5. Firewall installation. By default, the firewall available on the Raspbian operating system is not configured and must be set up by the user.

### 5.5.2 Firewall Protection

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).

### 5.5.3 MQTT Communication Security

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 24, 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.

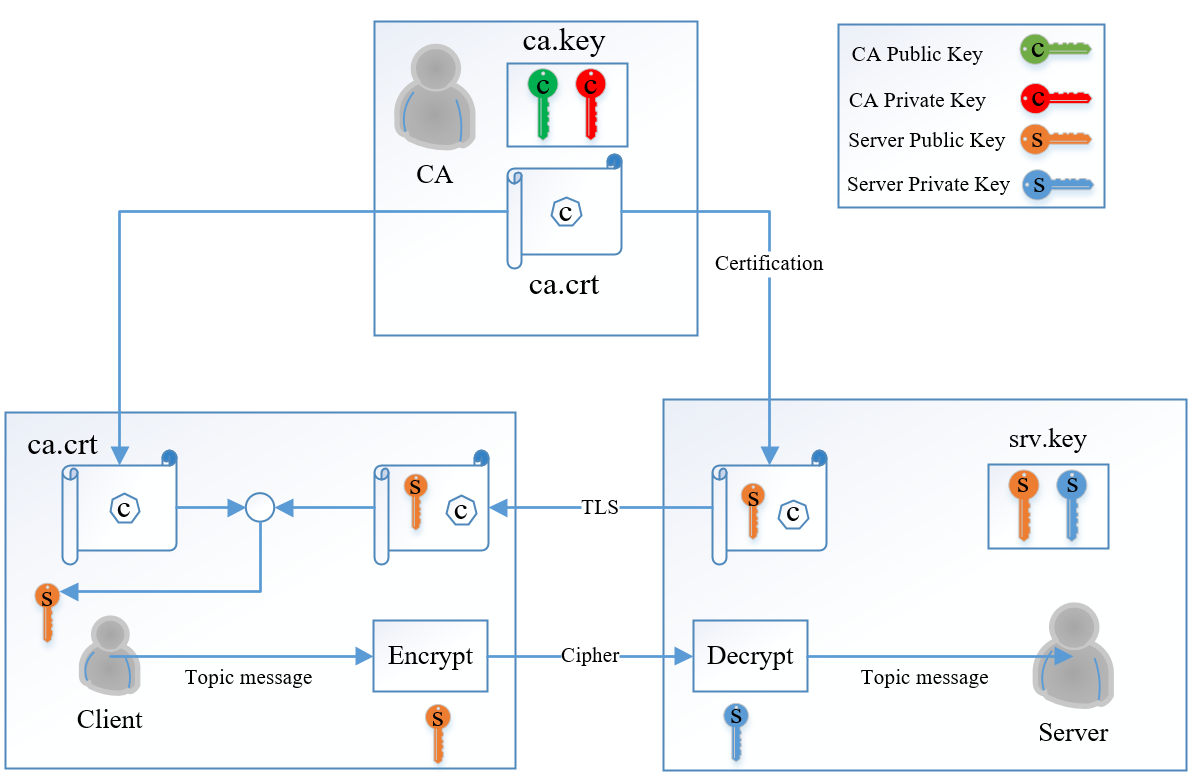


Figure 24: MQTT Communication secured using Certification and TLS

# 6.0 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 Pi), portability and determining how secure the model actually is.

## 6.1 Surveillance System Performance

### 6.1.1 Accuracy of Algorithms

To evaluate the accuracy of the system, multiple false-negative tests have been set up, meaning the classifier will attempt to accurately predict the individual it is trained on when they are present in sight of the camera. Both the Histogram of Gradients and Haar-Cascades algorithms are evaluated. If the classifier predicts the individual as unknown or another person, this can be considered a false negative. Figure 25 shows the contrast between these two results visually. 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 less false positives occur and thus higher accuracy achieved by the classifier.

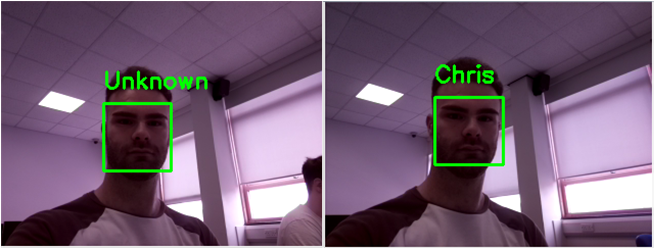


Figure 25: Difference between a false negative and true positive result in facial recognition

The results shown in Table 12 and Table 13 highlight a noticeable disparity between the two algorithms in terms of accuracy. It appears that the HOG algorithm is more accurate than the Haar-Cascades method of face recognition regardless of the images trained on the model. The data shows that regardless of the algorithm, classifier accuracy almost always increases with the number of encoded images, with diminishing returns after five images encoded. It can be concluded that the HOG algorithm is the most accurate out of the two and is best used with at least five images of the individual that wishes to be recognised. It should be highlighted here that the sample size for each experiment is low and therefore is not a true reflection of these classifiers, but rather a general guideline.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Images Encoded | True Positive | False Negative | Accuracy | Sample Size |
| 1 | 68 | 32 | 68% | 100 |
| 5 | 78 | 22 | 78% | 100 |
| 10 | 77 | 23 | 77% | 100 |

Table 12: HOG Algorithm Accuracy recorded using different volumes of encoded images

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Images Encoded | True Positive | False Negative | Accuracy | Sample Size |
| 1 | 50 | 50 | 50% | 100 |
| 5 | 55 | 45 | 55% | 100 |
| 10 | 59 | 50 | 59% | 100 |

Table 13: Haar-Cascades Algorithm Accuracy using different volumes of encoded images

### 6.1.2 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 Histogram of Oriented Gradients (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 26, 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 26: Difference in Computational Stress between the HOG and Haar algorithms

### 6.1.3 Memory Consumption

To further evaluate the performance of the algorithm, memory consumption was used as an additional variable. As shown in Figure 27, it is clear that the Haar-Cascades algorithm consumes significantly more memory than the HOG algorithm, with it consistently consuming more memory.



Figure 27: Difference in Memory Consumption between the HOG and Haar algorithms using 10 encoded images

Table 14 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 14: Comparison of CPU & Memory usage between the HOG & Haar algorithms

## 6.2 System Latency

To evaluate the latency of the system, data must be recorded via timestamp utilisation. This involves generating a timestamp at the exact time an individual is picked up on the camera and an additional timestamp is created as soon as a push notification occurs on the user’s phone.

## 6.3 Portability

## 6.4 Security

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 |  |
| 2.4.5.2 |  |
| 2.4.5.5 |  |
| 2.4.5.6 |  |
| 2.4.5.8 |  |
| 2.4.5.21 | The device will utilise MQTT for communication & use cryptography outlined in 2.5.3.1 |
| 2.4.6.1 |  |
| 2.4.6.3 |  |
| 2.4.6.4 |  |
| 2.4.6.5 |  |
| 2.4.6.6 |  |
| 2.4.6.7 |  |
| 2.4.6.8 |  |
| 2.4.6.9 |  |
| 2.4.6.10 |  |
| 2.4.6.11 |  |
| 2.4.7.1 |  |
| 2.4.7.4 |  |
| 2.4.7.5 |  |
| 2.4.7.6 |  |
| 2.4.7.10 |  |
| 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 15: Proposed solutions for IoT Security best practices

# 7.0 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].