Security in IOT Devices

Final report

Submitted for the BSc in

Computer Science

April 2021

by

Kai Christopher Michael Tindall

Word count: XXXXX

This document is a template for your guidance. You don’t have to stick to it precisely. It may not suit your particular project. Modify it if you need to – but please discuss with your supervisor before making significant changes to the organisation and content.

Paragraphs in red, like this one, are instructions and extra information. You must delete them before submitting your report.

This template document has a number of paragraph styles predefined. If you use them (‘Heading 1’, ‘Heading 2’ and ‘Heading 3’) and don’t alter them, then your report will automatically have properly numbered paragraphs and your table of contents will be automatically generated with the right page numbers. Use ‘Normal’ as the style for general text paragraphs in your document.

The document is divided into sections by ‘section breaks’ in Word. These are needed to keep the page numbering correct, so only delete them if you know what you are doing.

On this page (and I hope this is obvious) you must replace the words “The Title Of Your Project”, “BSc/MEng”, “Your degree title here” and “Your Full Name Here” with the correct information. You’d be surprised how often people forget to do this.

You must also replace “XXXXX” with the actual word count (excluding acknowledgements, abstract, table of contents, references and appendices) of your document. Note penalties for exceeding the word count will be applied, according to university regulations.

Abstract

The abstract is a short, self-contained statement describing the whole of your work. It should be less than a page (typically half a page) and should summarise the scope, purpose, results and content of the work. The abstract might be thought of as a summary which you would read quickly to decide if the rest of the document is worth taking the time to read in detail. In scientific publishing, abstracts are often used as sources of keywords and concepts for searching, so it’s important to ensure that the main ideas and conclusions of your work are present.

When someone has read your abstract, they should know what your project was about, how you did it and what the end result was. It doesn’t need to contain references or literature reviews. For example, here’s the abstract from Bell and Brooks (2019)’ s paper “*What makes students satisfied? A discussion and analysis of the UK's national student survey*” (which, incidentally, is very interesting):

“*This paper analyses data from the National Students Survey, determining which groups of students expressed the greatest levels of satisfaction. We find students registered on clinical degrees and those studying humanities to be the most satisfied, with those in general engineering and media studies the least. We also find contentment to be higher among part-time students, and significantly higher among Russell group and post-1992 universities. We further investigate the sub-areas that drive overall student satisfaction, finding teaching and course organisation to be the most important aspects, with resources and assessment and feedback far less relevant. We then develop a multi-attribute measure of satisfaction which we argue produces a more accurate and more stable reflection of overall student satisfaction than that based on a single question.*”

Note that it’s very specific. There is no waffle, just details of what they did, how they did it, and what their conclusions were.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

This project creates a secure environment for IoT devices to transmit their data to a central control server. This server then publishes that information to an API that the end use can access. I did this using the open-source Crypto++ library for the cryptological functionality. TALK ABOUT RESULTS WHEN AVAILABLE. IF USED HTTP AS API THEN TALK ABOUT LIBS USED FOR THAT.

Acknowledgements

*I would like to acknowledge the support of the Crypto++ Library contributors for their useful online resources and the free and open nature of their project. Without them this project would have taken significantly longer.*

Contents

[Abstract i](#_Toc31119449)

[Acknowledgements ii](#_Toc31119450)

[1 Introduction 3](#_Toc31119451)

[1.1 Background to the project 3](#_Toc31119452)

[1.2 Aims and objectives 3](#_Toc31119453)

[1.3 Research question 3](#_Toc31119454)

[2 Literature review 4](#_Toc31119455)

[3 Requirements 5](#_Toc31119456)

[3.1 Product requirements 5](#_Toc31119457)

[3.2 Functional requirements 5](#_Toc31119458)

[3.2.1 Interfaces 5](#_Toc31119459)

[3.2.2 Functional Capabilities 5](#_Toc31119460)

[3.2.3 Performance Levels 5](#_Toc31119461)

[3.2.4 Data Structures/Elements 5](#_Toc31119462)

[3.2.5 Safety 5](#_Toc31119463)

[3.2.6 Reliability 5](#_Toc31119464)

[3.2.7 Security/Privacy 5](#_Toc31119465)

[3.2.8 Quality 5](#_Toc31119466)

[3.2.9 Constraints and Limitations 5](#_Toc31119467)

[3.2.10 Performance requirements 5](#_Toc31119468)

[3.3 Design constraints 5](#_Toc31119469)

[4 Design 6](#_Toc31119470)

[4.1 Software design 6](#_Toc31119471)

[4.2 Experimental design 6](#_Toc31119472)

[5 Implementation and testing 7](#_Toc31119473)

[5.1 Implementation 7](#_Toc31119474)

[5.2 Testing 7](#_Toc31119475)

[6 Evaluation 8](#_Toc31119476)

[7 Conclusion 9](#_Toc31119477)

[References 10](#_Toc31119478)

[Appendix A – Interesting but not vital material 11](#_Toc31119479)

[Appendix B – Other things which may be useful 12](#_Toc31119480)

# Introduction

In this section you will describe the project’s purpose, aims and objectives. You will introduce the project’s stakeholders and the reason for doing it. You will provide a brief overview of the report’s organisation – which may of course be different to this template.

There may be some overlap with the content of the PID in this section, but it should not simply be a repeat. The introduction in this report will be informed by the activities you have undertaken and their results, whereas the PID was concerned with forward planning.

Note that the sub-headings below are suggestions only; you may organise this section differently as appropriate to your project.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

## Background to the project

With the availability of cheap single board computers being at an all time high, more people than ever are turning to them to create their own smart home devices. With these DIY smart home devices, most people have two options: Integrate the solution with an open-source platform like Home Assistant, or don’t and check on the status of the device manually. Both options aren’t particularly great. The average hobbyist will struggle to modify their solution to work with a solution like Home Assistant, and just leaving their project and checking the status manually becomes worse the more devices you need to check.

## Aims and objectives

The project’s aim was to create a system to allow users to monitor the status of multiple IoT devices in an easy-to-use manner.

The project also had some primary and secondary objectives.

|  |  |
| --- | --- |
| **Primary Objective** | **Justification** |
| To create a driver software for the IoT devices to be able to talk to the command and control server. | The driver software allows the IoT devices to communicate to the command and control server securely using encryption. This is needed for data to not be usable if intercepted in transit by a hacker. |
| To create a command and control server to manage and communicates to the various IoT devices. | By having the command and control server do all the management of the IoT devices it lightens the load of the client, which means it will then be “dumber” and therefore easier to develop for end users. |
| To create and document an API that is exposed by the command and control server. | Using an API to transmit and receive commands and data between the client and the command and control server means that each implementation of the client doesn’t need to make sure it handles the business logic correctly and means that data will not be different between clients. |
| To create an example client that will communicate to the command and control server via the API. | This will allow users to be able to see and use the system without having to invest time developing a client for their chosen platform first. |

|  |  |
| --- | --- |
| **Secondary Objective** | **Justification** |
| To create three interchangeable secure communication methods that the user can choose from within the client. | This will then allow the user to pick and choose how they want their data to be encrypted depending on their individual circumstances. |
| Evaluate the best method of encryption for speed. | This will then allow me to recommend the chosen method to users who need a lot of data encrypted, but don’t necessarily need it to be the most secure. |
| Evaluate the best method of encryption for security. | This will then allow me to recommend the chosen method to users who need the highest level of security available for their data and don’t necessarily care if it makes it a bit slower. |

# Literature review

The literature review is a survey of the history and state of the art in the domain of your project. It will summarize the work that has already been done in the field; this may be scientific literature, known techniques, and even previous student projects. It will provide a historical perspective on how the subject area has arrived at its current state by looking at important developments over time. If appropriate, it may examine existing software in the domain, especially in terms of the technology used and the features offered. The focus of the literature review is to summarise the existing arguments and ideas of others, identifying which are important.

A good literature review could be a project on its own, and form a very useful guide to anyone new to the particular field. It would identify the important work, authors and publications which would be a good place to begin research activity. Open questions and areas where new work is required would be discussed. Really good reviews are often published in scientific journals. Your literature review is not expected to be quite so substantial, but should still provide a comprehensive summary which will allow the reader to understand the field.

Images can be very useful here. Remember to attribute them properly to avoid accusations of plagiarism. Your literature review will naturally refer to lots of existing work, which must all be properly cited and referenced – see the ‘References’ section towards the end of this document.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

# Requirements

## IoT Device Requirements

|  |  |
| --- | --- |
| **Requirement ID** | **Description** |
| SITD-DD-001 | The IoT device shall run the driver software on start-up. |
| SITD-DD-002 | The IoT device shall be able to transmit messages to the C&C server. |
| SITD-DD-003 | The IoT device shall be able to receive message from the C&C server. |
| SITD-DD-004 | The IoT device shall be able to decode encrypted messages. |
| SITD-DD-005 | The IoT device shall be able to encode messages. |
| SITD-DD-006 | The IoT device shall be able to sign messages. |
| SITD-DD-017 | The IoT device shall be able to check the authenticity of messages. |
| SITD-DD-007 | The IoT device shall be able to perform a key exchange with the C&C server. |
| SITD-DD-008 | The IoT device shall reject unauthorised communication. |
| SITD-DD-009 | The IoT device shall not crash if it encounters erroneous traffic. |
| SITD-DD-010 | The IoT device shall reply with a standard error report when it encounters erroneous traffic. |
| SITD-DD-011 | The IoT device shall reply to any authorised traffic with an acknowledgement message. |
| SITD-DD-012 | The IoT device shall wait for an acknowledgement message before sending repeat messages. |
| SITD-DD-013 | The IoT device should wait for 100 milliseconds before attempting repeat messages. |
| SITD-DD-014 | The IoT device shall be able to receive a workload to be deployed. |
| SITD-DD-015 | The IoT device shall be able to receive data to pass through to the running workload. |
| SITD-DD-016 | The IoT device shall be able to transmit data from the running workload back to the command and control server. |
| SITD-DD-017 | The IoT device shall receive the encryption method being used by the C&C server. |

## Command and Control Server Requirements

|  |  |
| --- | --- |
| **Requirement ID** | **Description** |
| SITD-CC-001 | The C&C server shall run on start-up. |
| SITD-CC-002 | The C&C server shall poll the network to discover new devices to connect to. |
| SITD-CC-003 | The C&C server should poll the network every 10 seconds. |
| SITD-CC-004 | The C&C server shall attempt to connect to a device when it receives the command to from the client. |
| SITD-CC-005 | The C&C server shall expose an API that the client can connect to. |
| SITD-CC-006 | The C&C server shall be able to encrypt traffic. |
| SITD-CC-007 | The C&C server shall be able to unencrypt traffic |
| SITD-CC-008 | The C&C server shall be able to perform a key exchange. |
| SITD-CC-009 | The C&C server shall be able to sign messages. |
| SITD-CC-010 | The C&C server shall be able to check the authenticity of messages. |
| SITD-CC-011 | The C&C server shall be able to transmit encrypted traffic. |
| SITD-CC-012 | The C&C server shall be able to receive encrypted traffic. |
| SITD-CC-013 | The C&C server shall reject unauthorised communication. |
| SITD-CC-014 | The C&C server shall not crash if it encounters erroneous traffic. |
| SITD-CC-015 | The C&C server shall serve API requests it receives from clients. |
| SITD-CC-016 | The C&C server may have the option to change the encryption method. |
| SITD-CC-017 | The C&C server shall act as a central authority for public keys. |

## Client Requirements

|  |  |
| --- | --- |
| **Requirement ID** | **Description** |
| SITD-CL-001 | The Client shall be able to perform a key exchange with the C&C server. |
| SITD-CL-002 | The Client shall be able to display information received from the C&C server. |
| SITD-CL-003 | The Client shall allow the user to input data to send to an IoT device. |
| SITD-CL-004 | The Client shall allow the user to send a workload to an IoT device. |
| SITD-CL-005 | The Client shall have a GUI. |
| SITD-CL-006 | The Client shall be able to sign encrypted traffic. |
| SITD-CL-007 | The Client shall be able to encrypt traffic. |
| SITD-CL-008 | The Client shall be able to decrypt traffic. |
| SITD-CL-009 | The Client shall be able to send encrypted traffic over an API. |
| SITD-CL-010 | The Client shall be able to receive encrypted traffic over an API. |
| SITD-CL-011 | The Client shall request data from the C&C server. |
| SITD-CL-012 | The Client should request data from the C&C server every five seconds. |
| SITD-CL-013 | The Client should give the user an option to change the amount of time the client waits to request data from the C&C server. |
| SITD-CL-014 | The Client shall reject unauthorised communication. |
| SITD-CL-015 | The Client shall not crash if it encounters erroneous traffic. |
|  |  |

## Security Requirements

|  |  |
| --- | --- |
| **Requirement ID** | **Description** |
| SITD-SEC-001 | All traffic shall be encrypted using a reputable method. |
| SITD-SEC-002 | All encryption methods shall be written by qualified reputable external authors. |
| SITD-SEC-003 | Each message shall be signed by the author. |
| SITD-SEC-004 | Encryption functionality shall be written as a library to enable reuse. |

# Design

If your project involves the development of software and/or hardware, then you will need to include a section in which you describe its design in detail. If you conduct any experiments (either in a research-oriented project or simply doing user evaluation) then you should describe their design and methodology here.

Delete the red paragraph and replace this one with your content (use the “Normal” paragraph style).

## Software design

### System Design

Diagram

Description automatically generated

Figure System Design

The overall system design shows the flow of data through the system and how it’s getting there. Most of the time internally (behind the API) all communication will be through encrypted TCP communications. The only time the communications will not be encrypted is when programs are exchanging keys to set up an encrypted line.

The API is the main method of communication into the system for the client. This will be in the form of an HTTP API, this allows the client to take almost any form, including a website.

### Security Library

The pivotal component to the project was the security library. This library needed to be designed with interfaces in mind to ensure that other encryption methods could be implemented and used quickly where the users needed it.

In order to achieve this without having to change other software to adapt to these changes, interfaces were designed along with a factor so that a configuration could be passed in and the appropriately configured service could be received regardless of what methods the user wished to use.

This library is intended to be used as a wrapper to the Crypto++ library, making it easier for the other programs to use security features without having to alter the code base to switch the type of method it uses.

A configuration generator was also included with file input/output options so that the user will be able to write their own configuration in whatever file format there is a matching file input/output method for. The file input/output class is also based on an interface so can be swapped for a different method very easily, should it need to be.

Diagram

Description automatically generated

Figure : Security Library Class Diagram

### Message Library

The message library needs to exist so that each program has a reference to how the data coming through is organised. There also needs to be some generic messages within the library otherwise it will result in a lot of bloat for every single use case of a message.

The StartTrans message is a special message sent from the device driver to start the transaction, this gives the server a chance to send back any special initial values that it may need to.

The rest of the key exchange procedure is done using the request and response messages. This is agreeing a symmetrical key to be able to use EncryptedMessages. Once a symmetrical key is agreed, then EncryptedMessages can start being used. The device driver uses these messages to send regular status updates mainly, and the server uses these messages to send commands like sending over a new workload.

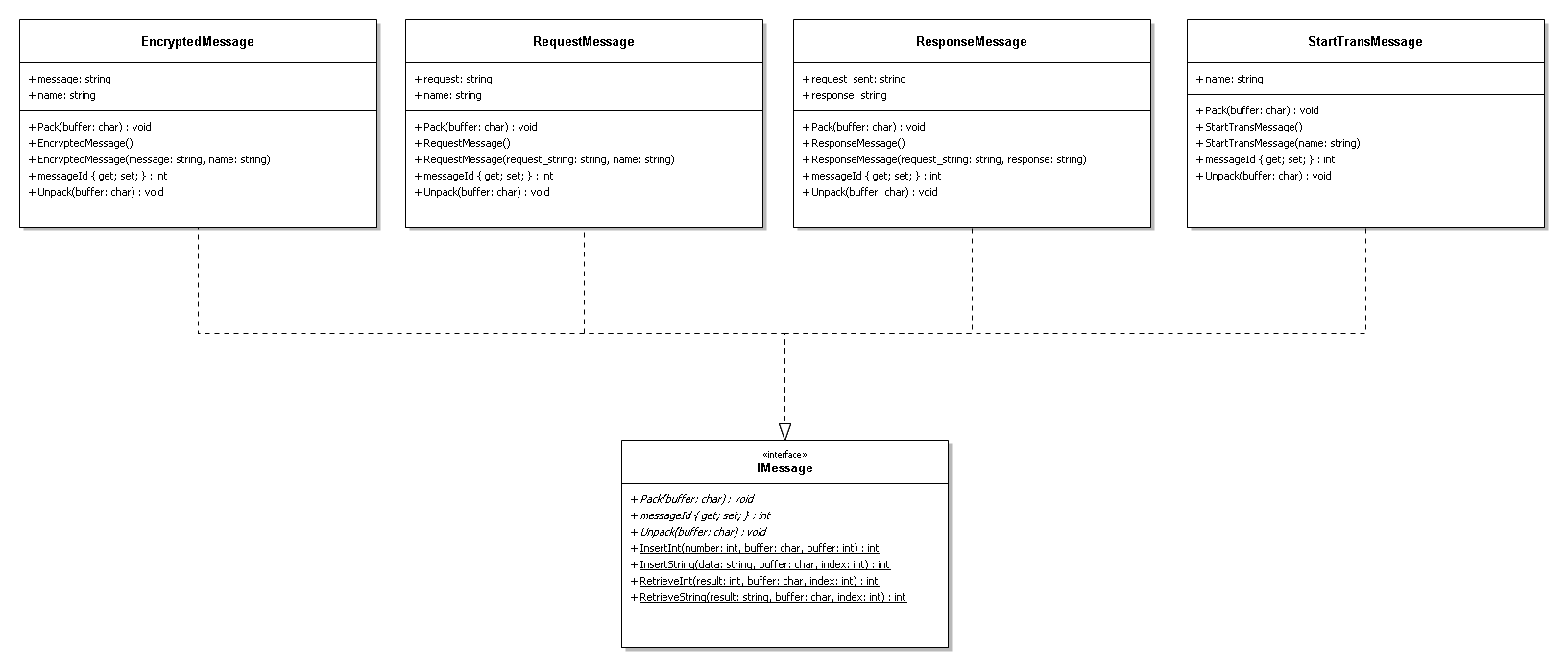


Figure Message Library Class Diagram

### Command and Control Server

The command server binds everything together and so is very important. However, it is also quite simple in concept. It only needs to handle connections coming through to it, which the actual connection can be handled with a separate class, and then just provide responses to requests.

There also needs to be some control functions for the main function to utilise. These are such as the IsRunning function which allows the main function to create a while loop and while the command server doesn’t receive a stop order, that while loop shall continue to receive incoming sockets and pass them through to the command server to handle in a separate thread.

Graphical user interface

Description automatically generated with medium confidence

Figure Command Server Class Diagram

When the command server sends out an encrypted message, the command server will encrypt and sign the data. This provides the trust that it has came from the command server with the signature, and the security that nobody else has seen it through using that encryption.

Diagram

Description automatically generated

Figure Command Server Message Sending Sequence Diagram

Diagram

Description automatically generated

Figure Providing Authentication Through Signatures

### Device Driver

The device driver is the software that will run on the IoT device. For the device to remain as responsive as possible, the device includes a message queue that different threads on the device driver can add to that will be processed continually by a sending thread. The message queue will also need a mutual exclusion lock to ensure synchronisation between threads, but this was left from the design for brevity.

Table

Description automatically generated

Figure Device Driver Class Diagram

### Public Information Registrar

The public information registrar acts as a central authority, similar to a certificate authority that’s used in web technologies. This tries to prevent an attack where the malicious actor tries to pass off their own public key as someone else’s. In the event they are successful, it means they can act as a man-in-the-middle and reveal information very easily.

Any program can connect to this registrar, and the registrar handles their request to either register a public key or request the public key with some filter parameters. Each program should have the public key of the registrar given to them so that the same attack could not happen where a malicious actor doesn’t pretend to be the registrar. This way the project can ensure that connections are genuinely between the correct parties.

A picture containing table

Description automatically generated

Figure Public Information Registrar Class Diagram

### Client

The client is the users main access point into the system and needs to gather all the information it can from the command server. Because the client is a graphical application too there is a graphics handler class that will deal with everything that needs to be displayed. The graphics handler has a data store object too that stores all the information that the graphics handler uses when it’s displaying the screen.

The API handler is a class specifically used to talk to the command server. When it gets the information back from the command server, it will parse it into the device data structure that represents the appropriate devices and update the datastore, which will in turn update what gets displayed onto the screen.

Diagram

Description automatically generated

Figure Client Class Diagram

This mock-up wireframe of the client illustrates how the client will look. This will be handled by the graphics handler class and all the status, workload, and defined commands will come from the datastore.

The aim of this design was to be quite minimal, not trying to add too much for the user, as to not over complicate it. All the devices the command server knows about is listed on the left, and if the user was to click on one of them it brings up the information panel about it. The information panel tells the user the devices current status, shows what workload the device is running and a button to upload a new one, and a section to send off predefined commands.

A picture containing diagram

Description automatically generated

Figure Client User Interface Wireframe

## API Design

### GET Requests

GET requests are HTTP requests that aim to get data from the server.

|  |  |  |
| --- | --- | --- |
| **Request Name** | **Parameters** | **Description** |
| GetServerPubKey | N/A | Gets the server’s public key from the public information registrar. |
| GetDeviceNames | Encyption Key | Gets the server’s list of devices connected. The client sends over their symmetrical encryption key, encrypted with the server’s public key. And the server sends back the data encrypted with the client’s key. |
| GetDeviceStatus | Encryption Key Device Name | Gets the status of a single device from the server. Request is encrypted with server’s public key and server encrypts response with encryption key provided. |

### `POST Requests

POST requests are HTTP requests that aim to give data to the server.

|  |  |  |
| --- | --- | --- |
| **Request Name** | **Parameters** | **Description** |
| PostDeviceCommand | Device Name Device Command Command Parameters | Tells the server to execute a predefined command on the device. |

## Experimental design

If you are going to evaluate your software or hardware by means of any tests or surveys, then explain their design here. If you are doing other experiments (for example measuring the performance of algorithms, extracting data from environmental monitoring systems or evaluating the performance of mechanisms) then you should explain how you have designed the experiments, how they must be conducted and what you expect to learn from them. This is especially important for research-oriented projects.

Delete the red paragraph and replace this one with your content (use the “Normal” paragraph style).

# Implementation and testing

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

## Implementation

In this section you will describe what you did, and why you made the important decisions affecting your actions. It’s not a diary – don’t write a blow-by-blow account of every little thing that happened. Be selective and report those choices and techniques which made a difference. Make sure you discuss what options you considered. Explain how the criteria and methodology you used to select amongst different options (which tools are most appropriate, for example).

It may help to imagine that you are reading this project in the future, trying to replicate the work without making the same mistakes along the way. What would you need to know to make your job easier, and what is unimportant or obvious? Explain how you implemented the design in the previous chapter.

This is the place in which you would explain any novel or especially complex algorithms, data structures or systems you have used.

Make it clear what you have done, and what is pre-existing. For example, if you are using third party software libraries, describe how you have used them, and how they have benefited your project rather than simply what they do. If you have built on a framework, make it clear how you have developed new functionality.

First, C++ was chosen as the language the project was to be written in. C++ was chosen because being cross-compatible on as many platforms as possible was very desirable due to not knowing what kind of platform the user would want to use the project on. Therefore, C++ was chosen over something like C#, which is a more modern language, but it’s only viable on Windows platforms. C++ was chosen over C because it has a lot more modern features like strings, classes, and namespaces. C++ can also use C functions very easily too so if needed be, C could be mixed in.

Now the language was decided, the project needed to be held in source control. Git was chosen to be the source control method used because it’s the most popular method (Slant, n.d.) and the project developers have extensive experience using it. GitHub was used as a remote repository because it’s free and has an extensive list of features like pull requests and actions. Competitors like GitLab were not considered because they’re a paid service.

Visual Studio code was chosen as the text editor to write the project in, this was chosen mainly for its lightweight nature. This was important since development was done in a Linux/GNU virtual machine, in order to be as close to the final target (Raspberry Pi) as possible. The other main option, Eclipse, was ruled out for this reason, because Eclipse is famous for it being very resource intensive.

Because one of the major reasons for picking C++ was to try to get the project to be very platform independent, CMake was chosen as a build configuration manager. CMake allows the project to define certain build parameters and when the user runs CMake it will compile the project using what’s best for the user, meaning the developers of the project don’t need to write complex make files.

Encryption was of course, a big part of the project. Because security is famously very difficult to write, and the best practice is to use a library (Cogswell, 2015), the project needed a library to use. Crypto++ was chosen as that library because of the extensive list of other projects that use Crypto++ (Crypto++, 2019).

Unit testing was decided to be used too, which also needed a library as it would take too much effort to write a custom framework specifically for this project. The googletest framework was chosen to be the unit testing framework used in the project. This framework was chosen over other frameworks such as the boost unit testing framework because the developers have had unpleasant times using the boost framework in the past, and the googletest framework was easy to use as it integrated with CMake well using the CTest feature, and was also easy to write.

Development was then started on the project. GitHub boards were used to assign work items, each time work started on a new work item a new branch was created to work on that specific item, and then when that work item was completed, a pull request was created that closed that specific work item.

Graphical user interface, text, application

Description automatically generated

Figure GitHub Boards Documentation Project

A picture containing text, monitor, screenshot, black

Description automatically generated

Figure GitHub Boards Closing Issues with Pull Requests

PugiXML was chosen to be the file IO library, this was since it seemed easy to use and was recommended by another developer who’d used it in a previous C++ project. The documentation was clear on how to use it, so it was implemented very easily and quickly.

The security library was implemented very close to the design and so was therefore very quick to write, although it did take longer than expected due to the complexity of the Crypto++ library. If this was to be done again, more time should be allocated when working with complex libraries.

The security library was thoroughly tested using automated unit testing through the google test framework. Every functional part of the library like the services had their own test suite dedicated to them, which could be easily expanded upon by just adding to the appropriate file.

By the time the security library was finished, the project was already quite far behind schedule. This was due to a few reasons, first designing the project was a lot more difficult than imagined, so a lot of time was used designing. This was due to the developer not having much experience designing software properly, so it was a big learning opportunity however used a lot more time than budgeted. Secondly, the schedule was very optimistic when it came to writing the implementation for the components, it took a lot longer than the week estimated per component. This should have been more obvious when planning due to the developer having limited experience with C++.

This led to the need to rescope the project. The decision was made to rescope the project to include a command server and device driver, with the public information registrar not being implemented currently but with it still in mind when writing the project. The client program was also left out unless there would be sufficient time to write an adequate one, with the replacement user interaction being a command line interface. The device driver would also for the time being only report fake statuses however would be written in such a fashion as to easily allow the device driver to report real statuses from a running workload.

The command server development began after the security library, with the decision made to write the message library as development went on as the programs needed the individual messages. A mix of reading documentation both from online sources, blogs, online tutorials, and command line ‘man’ (manual) pages on the relevant TCP functions led the development of the TCP server aspect of the command server with fast results. After that it was on to writing the command server class which utilised the TCP server class to receive incoming clients and pass them onto a command server function running within a separate thread. Threading was utilised within the server as without it only one connection could be held at one time, which is obviously not very good for a server.

A separate function was written in the command server to handle an incoming message from the connected socket. This function would use the static function ReceiveInt from the message library to extract the message ID of the incoming data, which could then be used within a switch statement and then unpack that raw data into the appropriate message structure.

The message library used the IMessage static functions to pack and unpack all the raw data. Everything consisted of integers and strings, and each message had a specific data order defined within the message class itself which put structure into the raw data. Integers were just stored as four ordinary bytes, but strings on the other hand needed to be put in a special sub-data-structure. The length of the string was first given as four bytes, which could then be used to read only the correct number of bytes which the string was within. This meant that the developer could easily insert and retrieve strings in raw byte arrays using two simple static functions.

When the command server successfully extracted a message, it would take an action defined in the handle message function, which, if it were a status message, would put the device’s status within an internal store that was shared across all threads. The message store was a map of strings to other maps, with the sub-map being a map of strings to strings which were the statuses of the device with the values. There was a single mutual exclusion lock for the entire data store, this means that the data store used coarse grain locking, which is fine for a few different threads trying to access it but will not scale well.

A better solution for this could be a finer grain of locking, which could be that each device’s status map could also be linked to a mutual exclusion lock, this would mean that if two separate devices are trying to update their status at the same time it wouldn’t matter. This improvement could massively improve individual status message turn around time in a bigger scale, say there’re ten thousand devices connected, each one of them in the current system needs to wait for the same mutual exclusion lock, whereas with the improvement they’d all have their own lock and would therefore dramatically reduce wait times.

The command line interface mentioned in the rescoping was implemented in its own thread, as not to disturb the command server in its purpose. The command line interface was given two commands, ‘List’ and ‘Show {Device}’, which would list the devices the command server currently had statuses for and would show the status of the device chosen. Thus, giving the user quick access to all device’s status from a single point.

The command server was tested throughout development using a telnet console. This allowed connections to be made and data to be easily sent before the device driver was created. Any issues with the command server after development had begun and progressed to a state where TCP packets were being transferred then was debugged using the prototype device driver.

The device driver software was to be written next and was frequently tested using the command server written. Some of the code from the TCP server written for the command server was recycled and used in the device driver, with the main difference being that instead of listening for connections it was creating them. The server was created with threading in mind, this allowed it to have multiple things happening at once that could each communicate with the command server. This also meant that a lot of synchronisation techniques were required to keep everything in line and prevent data races.

The most common, of course, were mutual exclusion locks. These are the most basic of synchronisation methods and a few were utilised to place critical sections where important data was accessed. Atomics were utilised extensively as flags for while loops in threads, as the client would need to be stopped gracefully if a premature exit were required. Atomics provided an easy way to achieve this as no locking of waiting needed to be used.

As mentioned above, when stopping prematurely the device driver would need to be stopped gracefully as to not cause long lasting port and IP problems for the device. Therefore, the SIGINT signal was intercepted in the code to instead of its normal behaviour, call stop on the device driver object.

The stop function would set the atomic flag to the false value to allow the threads to exit and then the main thread would wait for a join for all threads until closing the socket and then exiting the main program.

The most complex synchronisation method used by far was a conditional variable. A conditional variable can best be described as kind of a ‘smart lock’. It allows a thread to wait until a variable meets a certain criterion and the conditional variable has been notified. This was especially useful when a thread would send a specific request to the server through the message queue and need to wait on the response. An example of where this was used was within the key exchange thread where the thread needed to wait on the server sending over its public part of the key.

This is where the current implementation of the project currently stands. The device driver is a very strong base and only really requires the feature of getting statuses from a real program, which should be easy enough to achieve. The command server has a basic version of the API, albeit only in the command line form, so this would need expanding to instead of being commands inputted from the console be commands inputted over a HTTP connection. The public information registrar was never implemented however can be easily inserted into the current state of the project. The client was never implemented but due to the open nature of the client a suitable client could be whipped up quickly using a framework like C#’s WPF. However the security library is fully operational and able to be expanded with different methods incredibly easy without changing the source code in any other program.

## Testing

If you are developing software or hardware, you must test it. This section should explain how your work will be (or has been) tested.

You should have a test plan at the very least (full details of it and its results if required can go in an appendix). Ideally, you will have automated tests for any software you build. You will also define user acceptance tests, or something similar which can be used to determine whether your output meets the requirements stated earlier. Explain how and when the tests should be conducted.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

# Evaluation

This section evaluates the *software (or other artefact)* you have developed. You should compare it with the original specification and see how well it satisfies the requirements. You may wish to refer back to your aims and objectives at this point. You should report the results of user testing and a summary of feedback if that has been collected.

If you have done experiments, then the results of these should be reported and discussed here.

If you have involved people in doing user evaluations, that information should be include here.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

# Conclusion

In this section you should evaluate the *project* as a whole, and draw conclusions from the work you have done. Ask yourself what the project has achieved – what is its contribution? Has it met its initial aims and objectives? If not, why? How does the work you have done enhance the field in general? What has been learned from the project? If you have a well defined research question, has it been answered? What do the results mean?

You should also use this section to reflect on the *process* by which you undertook the project. Was your methodology appropriate (and did you stick to it)? Was your time planning good? Did you complete the primary and secondary objectives, and if not then why? What have you learned from the process? What would you do better/differently if you had more time?

Sometimes, it’s appropriate to include a subsection on ‘Further work’, making suggestions of how to proceed and what could be done to enhance the project in future.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

References

References must be formatted in the correct manner. For this assignment you must use the University of Hull’ approved variant of the Harvard referencing style (Fallin 2019), fully described at https://libguides.hull.ac.uk/referencing/harvard. Note that the details of the expected format vary depending on the type of document being referenced. Make sure you are familiar with them. If you use reference management software such as Zotero, EndNote or RefWorks, then you should be able to export a table of references in the correct format, which will save you work.

Every reference should have at least one citation in the text. Most will probably be in the ‘Background’ or ‘Literature review’ sections.

Remember that there is a difference between references and a bibliography. You will certainly need references, but a bibliography is optional.

There is much more information and guidance about referencing on the library’s website at https://libguides.hull.ac.uk/referencing/home

Some examples, illustrating different types of source:

Bahraini, M.S., Bozorg, M., Rad, A.B., (2018). SLAM in dynamic environments via ML-RANSAC. *Mechatronics* 49, 105–118.

Fallin, L., (2019)*. LibGuides: Referencing your work: Harvard Hull.* Available online: http://libguides.hull.ac.uk/referencing/harvard (accessed 10/10/2019).

Janis, I., (1972). *Victims of Groupthink: A psychological study of foreign-policy decisions and fiascoes.* Houghton Mifflin, Boston.

Office For Students (2018) *. Securing student success: Regulation framework for higher education in England*. Available online: https://www.officeforstudents.org.uk/media/1406/ofs2018\_01.pdf (accessed 10/10/2019)

Schmuck, P., Chli, M., (2019). CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams. *Journal of Field Robotics* 36, 763–781.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

Appendix A – Interesting but not vital material

Appendices are used to include information which may be of interest but is not necessary for the reader. You do not have to include appendices if there is no need for them.

You might, for example, want to include some details of a particular piece of software (an API, perhaps) or hardware which your project uses. This might be something that a reader might wish to consult, but you wouldn’t want to include in the main body of the report. You could also put raw data from experiments in an appendix, or perhaps survey results. It should still be information of relevance, but nothing that everyone would be expected to read.

If you wish to refer to elements of your PID, you could include them in appendices.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

Appendix B – Other things which may be useful

You can have more than one appendix, or none at all. Give them meaningful names and titles (not the ones given here), so that you can refer to them in the text, and so that they appear in the table of contents.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).