Hugh Wells 2020

An investigation into attendance monitoring systems and the Web Bluetooth API

Wells, Hugh

A report submitted in part fulfillment of the degree of
BSc (Hons) in Computer Science

Supervised by:
Professor Hugh Shanahan

Department of Computer Science
Royal Holloway, University of London

Word Count: 18,000 words

Student Name: Hugh Wells

Date of Submission: 23rd April 2020

Signature:

H Wells

Contents

Abstract

This project researches existing attendance monitoring solutions and conducts a security analysis of the existing solutions at Royal Holloway. It culminates in the design and build of an MVP (Minimum Viable Product) to test assumptions and demonstrate the core ideas of a proposed solution.

Introduction

Royal Holloway keeps track of attendance in lectures both to ensure that students are regularly attending and also to satisfy legal requirements regarding the visas of overseas students [1]. It is essential that this data is gathered and analysed efficiently and accurately.

In the past attendance has been tracked using signatures on registers. More recently, due to the lack of scalability of this former approach, a system of clickers has been employed [2]. This latter system is potentially non-optimal and insecure.

Outline

The project firstly the security issues with the current clicker system - looking at confidentiality, integrity and availability. Some tools to intercept messages generated by the clicker are demonstrated, and theoretical spoofing is explored.

Browser fingerprinting libraries are then demonstrated, and the ethics behind these techniques are later covered in as part of the Professional Issues section. Two solutions are identified and implemented.

An analysis of the communication technology is then included, looking firstly at MIFARE and latterly at the Bluetooth Low Energy specification and Web Bluetooth APIs. Alternatives and issues are discussed.

A new MVP and PoC (Proof of Concept) system for attendance monitoring is proposed in this project using ideas developed in the research phase, to solve issues identified with the previous solutions. This demonstrates key ideas, however is not a comprehensive solution.

Motivation

The primary motivation for this project is to offer a more secure and convenient system for students. The author is well acquainted with both the paper based and clicker systems in use, and the associated merits and pitfalls. An improvement in the process would allow less detractions from the lecture content but, as Universities UK writes: "Attendance monitoring can also be perceived as unfair and harm international student experience." [3]. Registers are often used for an entire cohort so as not to single out international students, but perhaps create the perception of international students imposing rules on domestic students as a result of their presence. A less onerous system would reduce the effort required by students and therefore by extension remove any perception of unfairness that may be harboured.

Universities UK continues, "The current system imposes a significant administrative burden on both institutions and the Home Office. . . " with a survey conducted by them concluding the the total cost of compliance with Tier 4 rules being £40 million to the UK Higher Education sector [3]. A separate study by EY[1] [4] for the Russell Group noted "Attendance monitoring is particularly time consuming across such a large university with many different modes of study. Collating the data, analysing it and escalating cases for investigation/explanation has created an industry of work for very little tangible benefit given that HEI students are very low risk of visa abuse.".

This work therefore will develop my skills in using hardware and building production applications with proper software engineering practices. I intend to explore my interest in the Bluetooth specification and the hardware elements of the project to provide context to the information security focus. As I intend to take a Masters in information security this research is directly relevant to my future goals.

[1]Ernst & Young, the accountancy and audit firm

Research

Literature Survey

In the research for the Clicker section, it was noted that there is very little research on this area and the main citations are for blog posts and code written by others. Later references are also informal technical guides. This was expected and has been previously noted in the Project Plan.

The browser fingerprinting research has two important citations: “User Tracking on the Web via Cross-Browser Fingerprinting” and “Device Fingerprinting: Analysis of Chosen Fingerprinting Methods”, the former of which presents the research related to the Fingerprint2 library also cited. These provide significant context, although other sources are cited for completeness and to give more practical examples (in the case of a newspaper article). This is also reflected in the research for the Professional Issues section, which shares topic and sources.

In the communication section of the research there are several areas of interest. When investigating the MIFARE Classic attack, the Mayes and Cid article has an excellent overview and provides additional sources which are also cited to give further context. The Nohl and Plötz video (later paper) is the original research and cited extensively. The Bluetooth Low Energy section is essentially based on the Bluetooth Specification, with support from two textbooks “Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking” and “Bluetooth Low Energy the Developer’s Handbook.”, the former is more theoretical and the latter more practical. A research paper is also cited but this refers to version 4.0 of the specification, and technology in 4.1 is current. The section on HM-10 investigations suffers from the same issue as the Clicker Report - as it is inherently practical, there are limited academic sources and indeed, the material out there at all is limited. Of particular interest will be the HM-10 specification (although in areas it is incorrect) plus the Web Bluetooth API specification.

As part of the design and research of the proof of concept solution, many sources of official documentation were consulted including those for Flask, PyMySQL and other Flask libraries. Alongside this textbooks such as “Flask in a Flask, I Mean, Book” by Perras added additional context and example code. A number of blogs talking about very specific technologies or concepts were used, but sparingly.

Clicker: reverse engineering

Overview

This section looks at the ways it is possible to attack the clicker system in use at Royal Holloway: the technology needed and the issues encountered when building upon existing research.

Introduction

At Royal Holloway, for the 2018 intake of first year undergraduates it was decided that due to the class size, paper registers were unfeasible. The clicker systems was proposed and developed. This uses a Turning Technologies “Response Card” (Figure 1) typically used to respond to interactive questionnaires as part of a slideshow, or with other proprietary software. The device communicates with a base station connected to a computer via USB (Universal Serial Bus) when a key option is pressed (eg. “1/A”) and transmits the unique ID of the device and the key press. The message is acknowledged by the base station and the user is given visual affirmation on the device that their response was counted. The results are then stored in a proprietary format attached to the slideshow which is decoded, processed and analysed by the department using a collection of scripts, Excel spreadsheets and custom software.

The aim was to investigate the Turning Point clickers to learn more about how they work in practice and to see to what extent it is possible to reverse engineer, intercept and spoof communications between the clicker and the basestation.

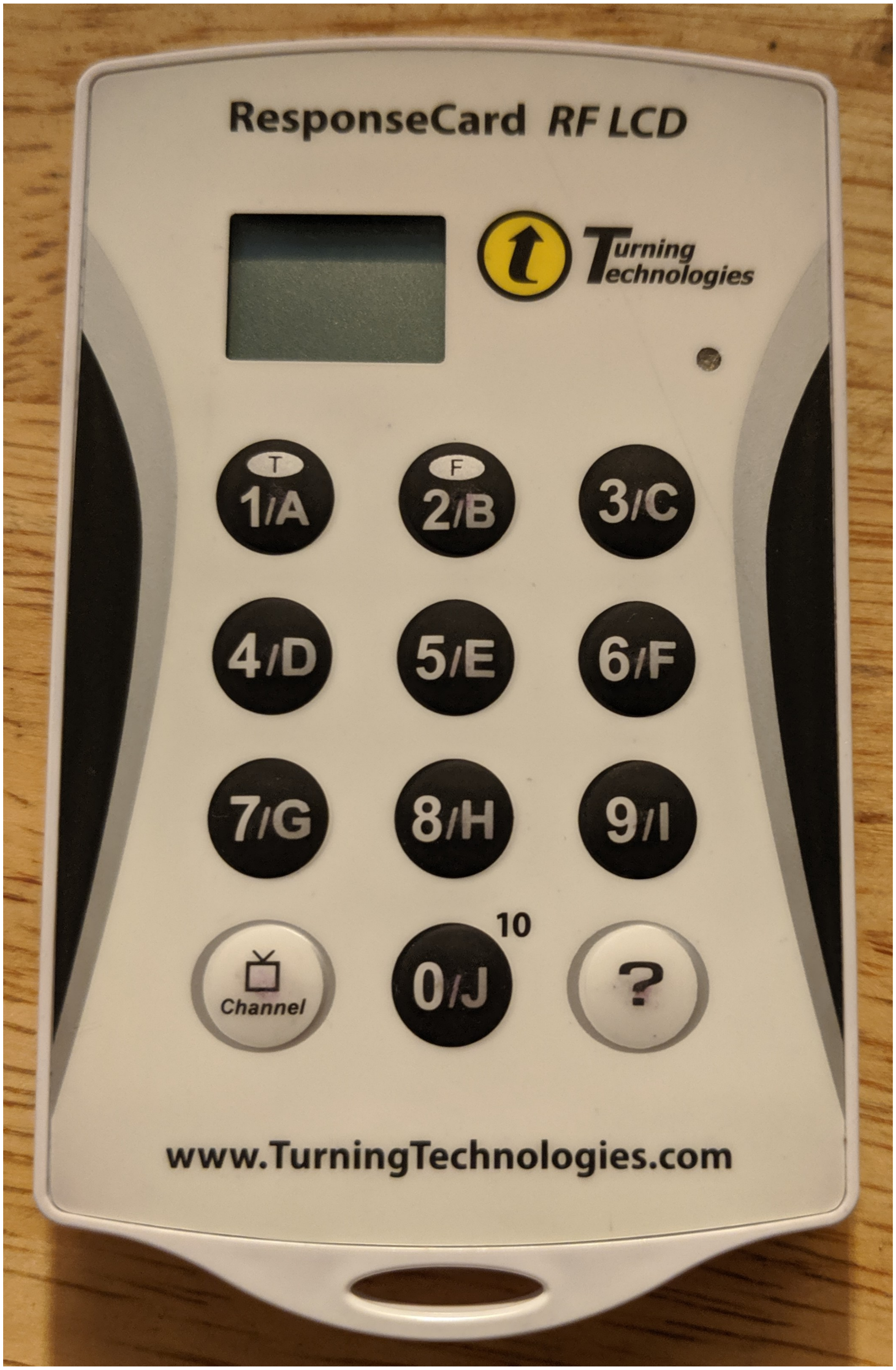


Figure 1: A Turning Technologies “Response Card”

Background Research

Whilst researching the device, I discovered the work of Travis Goodspeed who reverse engineered a similar, but older device. [5] This was achieved by dumping the firmware of the device allowing analysis of the way the device operated and how packets were structured and sent. A number of important discoveries were made, firstly that the System on Chip is a Nordic nRF24E1 chip, which is a Intel MCS-51 (8051 microcontroller) with an nRF2401 radio transceiver. The nRF2401 is a 2.4GHz, serial radio transceiver [6]. The datasheet lists a number of potential applications including telemetry, keyless entry and home security and automation. The chip also uses what the datasheet describes as a "3-wire serial interface." - this is otherwise know as SPI (Serial Peripheral Interface) and allows easy interface with devices such as a Raspberry Pi and Arduino. Secondly, there is no encryption and in the aforementioned research it was noted, through analysis of the firmware, that the packets take the structure of three bytes target MAC, 3 bytes source MAC and then a single byte for the button selection. A CRC is calculated and added by the radio.

This means that the source, destination and the parameter (the button pressed) are transmitted in cleartext with the only validation being a CRC (Cyclic Redundancy Check). The message is not signed, so there is no way to verify that a message has indeed come from the advertised source - this allows an attacker to arbitrarily spoof messages purporting to come from any source MAC address.

Cyclic Redundancy Check

CRCs are a method of error checking that are widely used in serial communications [7]. Traditionally, parity bits have been used to ensure data integrity across a communication channel. These indicate whether the data value is expected to be even or odd, but suffers from two major flaws. Firstly, it is not possible for you to determine where the corruption has ocurred in a piece of data - it is not possible to repair the message and it has to be discarded and resent. Secondly, if two corruptions occur then the data may end up passing the parity check but still be invalid. Additional measures (such as length checking, strict processing validation etc.) can be included to reduce the risks of bad data being processed. Whilst a parity bit is technically a 1 bit CRC (CRC-1), greater check values are now used as the greater bit value allows you to determine reliably how many bits have been corrupted and therefore protects against corruptions that fail to modify the overall parity.

CRCs work using modulo arithmetic on some generated polynomials. The general formula to calculate a CRC is:

$CRC = remainderof \left[M(x) \times \frac{x^n}{G(x)} \right]$

Where `M(x)` is the sum of the message used as the coefficients in a polynomial and `G(x)` is known as the "generating polynomial". This is generally defined by the type of CRC you are using: for CRC-16 (which is the type used for the nRF24E1) `G(x)` is defined as:

$G(x) = (1 \times x^{16}) + (0 \times x^{15}) + (1 \times x^2) + 1$

This produces a value (the polynomial) when is then used to divide `M(x)`, with the remainder used as the CRC.

In this particular situation, it is possible to calculate the CRC on the nRF24E1 in the ShockBurst™configuration settings by setting the `CRC_EN` flag bit. This means the chip will calculate and append the CRC as well as strip and validate the CRC on messages received [6]. In the Nick Mooney implementation [8], CRC is calculated on the Arduino not on the nRF24E1 due to technical issues getting this working. The CRC is instead calculated and checked using an alternative library [9].

Hardware

It is possible to use any Arduino and nRF24E1 breakout board for this project - I elected to use the Arduino Nano and the cheapest nRF24E1 board I could find on eBay. The Arduino is placed on a breadboard and the nRF24E1 connected via Dupont leads to the requisite pins.

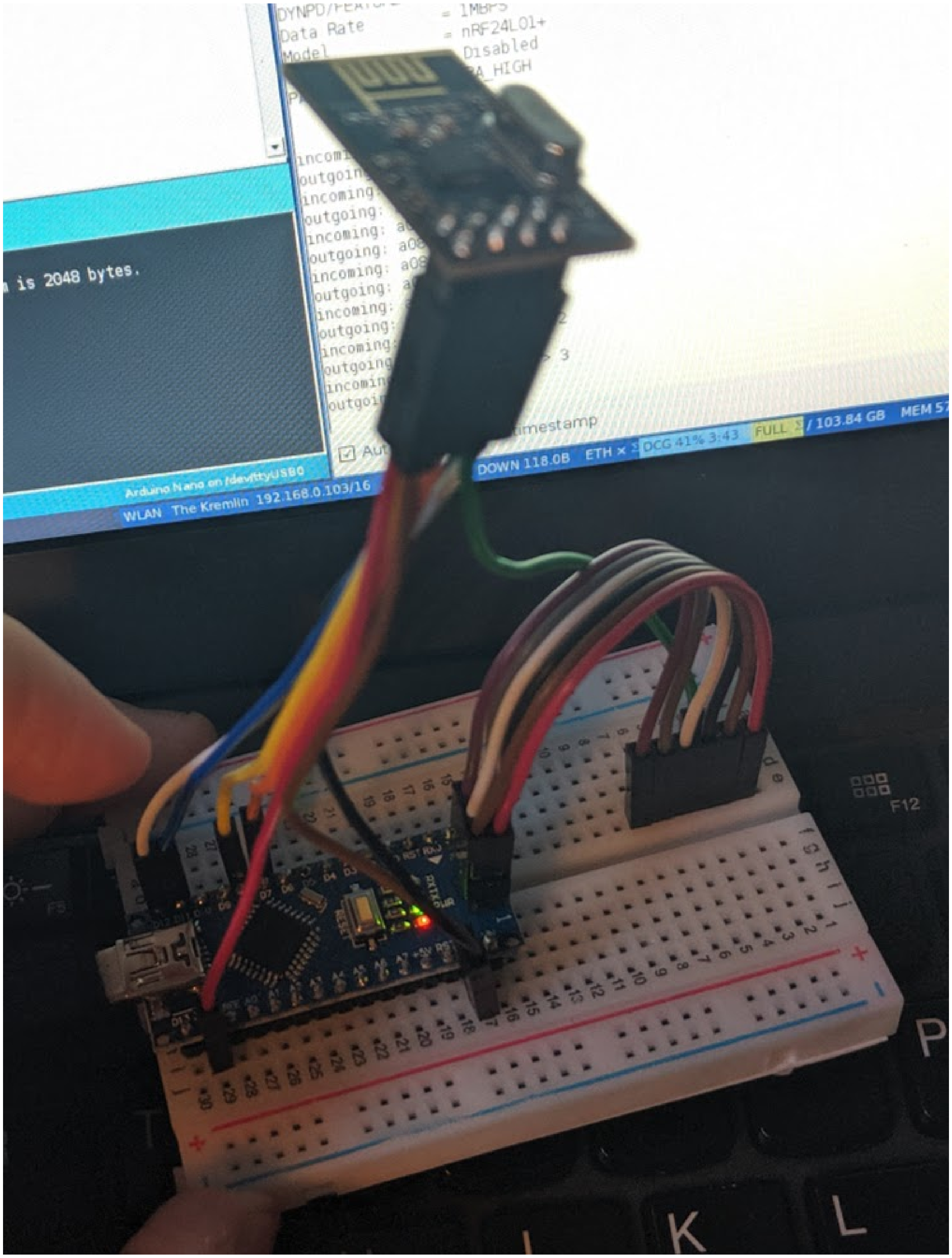


Figure 2: The wired up Arduino breadboard

As always, this can be a somewhat complex operation and it did require consultation of not only the nRF24E1's advertised pin layout[10] but also a setup guide for a similar, but not identical product from Sparkfun [11].

The pinout I used is listed in Table 1.

Table 1: The chosen pinout.

Name	nRF24E1	Arduino
VCC	2	3.3v
GND	1	GND
IRQ	8	GND
CE	3	D7
CSN	4	D8
MOSI	6	D11
MISO	7	D12
SCK	5	D13

With the addition of a mini USB cable to connect the Arduino to your computer, the hardware setup is complete. It is important to note that the hardware required for the base station emulator is identical to that required for the clicker emulator.

Clicker Basestation

The script to emulate the basestation of the clicker system is made up of two parts, the Arduino code and Python script. The former is essentially an unmodified version of the Nick Mooney file [8], which was developed in 2016. The interesting point here is that his work was developed on an entirely different device, and shows how widely this system has been deployed and the range of different devices available all using the same protocol [12]. The Arduino code listens for a transmission on the configured channel and then checks the CRC (discussed above). It then prints out the received packet to the serial console and returns the "accepted" message to the clicker. This causes the clicker to flash a green LED instead of a red one to show proper acknowledgement to a message by the basestation.

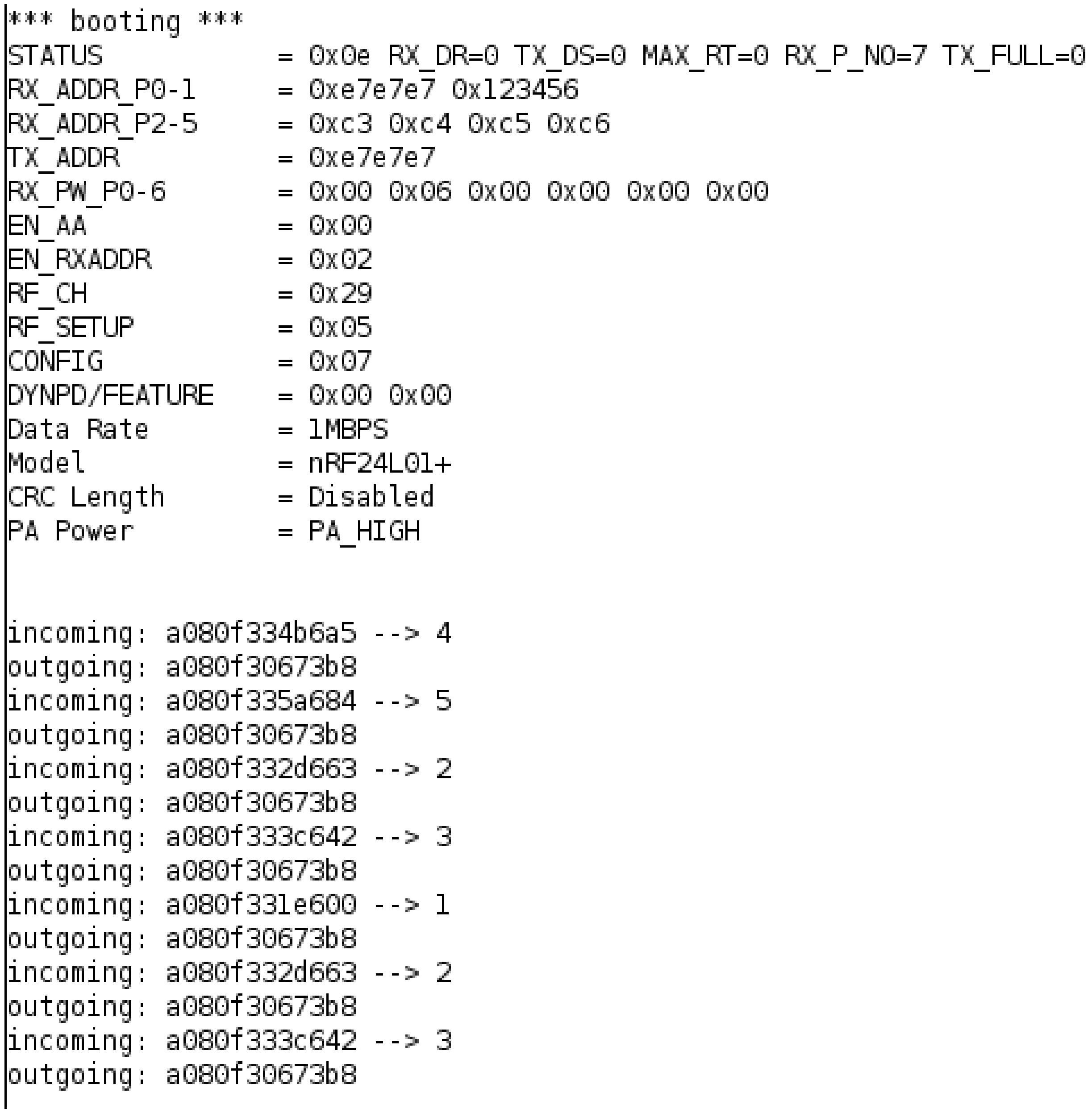
The `cli.py` script (Appendix 0) is substantially modified from the original Nick Mooney code [8] and provides a basic serial program to interact with the basestation emulator. It connects to the serial port and parses the aforementioned outputs, before saving them to a specified output file in a CSV format. Because of the way the serial printing is formatted, regex is required - the expression is the work of Nick Mooney.

An example output from the script:

```
address_from,address_to,button
a080f3,31e600,1
a080f3,35a684,5
a080f3,396708,9
a080f3,396708,9
a080f3,387729,8
a08167,35459b,5
a08167,35459b,5
a08167,35459b,5
a08167,35459b,5
a08167,35459b,5
a08167,35459b,5
a08167,35459b,5
a08167,35459b,5
```

As you can see, the `address_from` (the clicker address), the `address_to` (the hardcoded basestation address) and the button pressed are included. This is a significant improvement on the output from the current solution as the output is in a text file format (CSV) which does not require preprocessing from a proprietary format.

The output on the serial console is shown in Figure 3.



```
*** booting ***
STATUS           = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
RX_ADDR_P0-1     = 0xe7e7e7 0x123456
RX_ADDR_P2-5     = 0xc3 0xc4 0xc5 0xc6
TX_ADDR          = 0xe7e7e7
RX_PW_P0-6       = 0x00 0x06 0x00 0x00 0x00 0x00
EN_AA            = 0x00
EN_RXADDR        = 0x02
RF_CH            = 0x29
RF_SETUP         = 0x05
CONFIG           = 0x07
DYNPD/FEATURE    = 0x00 0x00
Data Rate        = 1MBPS
Model            = nRF24L01+
CRC Length       = Disabled
PA Power         = PA_HIGH


incoming: a080f334b6a5 --> 4
outgoing: a080f30673b8
incoming: a080f335a684 --> 5
outgoing: a080f30673b8
incoming: a080f332d663 --> 2
outgoing: a080f30673b8
incoming: a080f333c642 --> 3
outgoing: a080f30673b8
incoming: a080f331e600 --> 1
outgoing: a080f30673b8
incoming: a080f332d663 --> 2
outgoing: a080f30673b8
incoming: a080f333c642 --> 3
outgoing: a080f30673b8
```

Figure 3: Clicker basestation serial console

Clicker emulator

The hardware for the emulation of a clicker is exactly the same as that required for a basestation - the difference is purely in the code and associated processing of data. My high level plan was:

- Modify Arduino script to send to the hardcoded basestation address

- Modify Arduino script to take input of "sent from" addresses over serial
- Write a Python utility script to feed the Arduino with clicker addresses to emulate

I first decided to tackle getting addresses into the Arduino over a serial connection - it appears deceptively simple! I intially planned to use the `String` components of the Arduino `Serial` library - `Serial.readStringUntil` [13], for example. This had the advantage of avoiding having to worry about lower level buffer constructs and also allowed abstraction from having to handle reading from the Serial buffer until a certain point (end of string etc.). This did not work for a few reasons:

- The addresses being sent are actually binary values, usually represented in hexadecimal, and the String library cannot deal with this raw data.
- The "until" also proved to be somewhat unreliable and would not terminate necessarily on a line feed.

I discovered that if you cannot use the String abstraction, you have to handle raw `Chars` - these are individual bytes that you place into a buffer of suitable length. Arduinos use Harvard rather than Von Neumann CPU architectures so the data and system memory (RAM) are on different buses - this makes a buffer overflow exploit difficult, but not impossible. You do therefore need to be careful when handling buffers to ensure you do not deliberately overflow a buffer.

The basic code (encapsulated in a loop) is as follows:

```
if(Serial.available() > 0 &&
    counter != BUFSIZE){

    char incoming = Serial.read();

    if (incoming == '\r') {

    } else {
        incomingData[counter] = incoming;
        counter++;
    }
}
```

So, when the Serial port is available and the counter has not reached the buffer size:

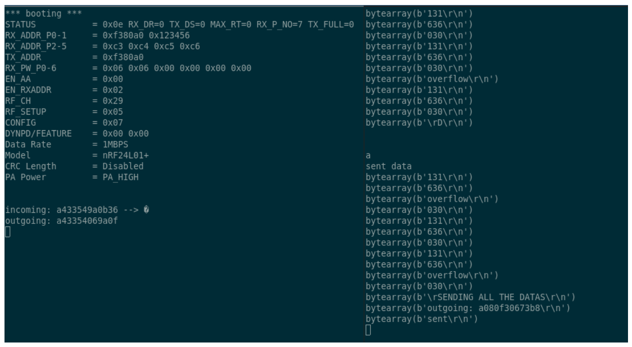
- Read a single character from the Serial port
- If it is a `\r` (line feed) then handle that case
- Otherwise, place it in the buffer and increment the counter

I discovered that you cannot just send `0xA0` - by doing so you are sending the literal ASCII codes for each individual char (eg. `A` = 65). Instead, you need to send the "hex values" for each byte, which is then a single char. One way to get a hex ASCII value is via `echo -ne "0xA0` which will print out the char you want. Sadly, when I sent this to the Arduino (using Screen) I ended up filling the buffer several times over with values that were completely unrelated. I still do not understand the cause of this issue but I suspect it may have been triggering some Screen escape codes.

I then moved to using a Python script which was more easily reproducible. In order to both read and send data over the Serial connection, I had two threads one to receive data and another to send characters as required. This was based on a StackOverflow post and used some example code, but the concept is not complex. In order to be able to both send and receive characters on a serial line in Python, you need at least two threads - one to process anything received on the serial line and print it to the console, and another to send data when instructed without blocking the receiving thread, as illustrated here:

```
conn.write(str.encode("160"))
conn.write(str.encode("160"))
conn.write(str.encode("160"))
conn.write(str.encode("\r"))
```

This more predictably populated the buffer at the other end, as shown in Figure 4. In the clicker basestation you can see one packet received - this is from other (presumably IoT) devices that use the same channel and addressing structure. You can also see the attempt to send data (on the right) with the various interesting echos back with different values to those which are sent, from the Arduino. The one outgoing packet shown to be sent was actually hardcoded and exposed another bug.



```
*** booting ***
STATUS           = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
RX_ADDR_P0-1     = 0xf380a0 0x123456
RX_ADDR_P2-5     = 0xc3 0xc4 0xc5 0xc6
TX_ADDR          = 0xf380a0
RX_PW_P0-6       = 0x06 0x06 0x00 0x00 0x00 0x00
EN_AA            = 0x00
EN_RXADDR        = 0x02
RF_CH            = 0x29
RF_SETUP         = 0x05
CONFIG           = 0x07
DYNPD/FEATURE    = 0x00 0x00
Data Rate        = 1MBPS
Model            = nRF24L01+
CRC Length       = Disabled
PA Power         = PA_HIGH


incoming: a433549a0b36 --> �
outgoing: a43354069a0f
```

```
bytearray(b'131\r\n')
bytearray(b'636\r\n')
bytearray(b'030\r\n')
bytearray(b'131\r\n')
bytearray(b'636\r\n')
bytearray(b'030\r\n')
bytearray(b'overflow\r\n')
bytearray(b'131\r\n')
bytearray(b'636\r\n')
bytearray(b'030\r\n')
bytearray(b'\rD\r\n')


a
sent data
bytearray(b'131\r\n')
bytearray(b'636\r\n')
bytearray(b'overflow\r\n')
bytearray(b'030\r\n')
bytearray(b'131\r\n')
bytearray(b'636\r\n')
bytearray(b'030\r\n')
bytearray(b'131\r\n')
bytearray(b'636\r\n')
bytearray(b'overflow\r\n')
bytearray(b'030\r\n')
bytearray(b'\rSENDING ALL THE DATAS\r\n')
bytearray(b'outgoing: a080f30673b8\r\n')
bytearray(b'sent\r\n')
```

Figure 4: Clicker basestation serial console on the left, clicker emulator Python Script on the right

It transpires that although the outgoing packet is correctly formatted (when compared to incoming packets from a clicker as received by the emulator basestation), no packet is actually sent or at least received at the other end. This is a fairly critical error and may stem from my misunderstanding over how the code works (which address is which) and issues with hexadecimal encoding between various components. It was not possible to rectify this given the time constraints and need to move on to other work.

Browser Fingerprinting

Overview

In order to ensure that students do not circumvent the registration process, it is important to build in appropriate security measures. If students share login details with others, it would be possible for those in attendance to log multiple people in on their single device. In order to prevent this attack we explore methods of identifying a device in the context of the chosen web technologies.

Introduction

Tracking a user within a website (or indeed across the internet) has been possible either by checking the IP address of the origin or by use of cookies for state management, the mechanism of which was described in the 1997 RFC [14]. As part of the new clicker system, it is envisioned that some method of identifying the device the student is signing in on will be required, to allow detection of one device signing multiple people in. The IP address is too general - those connected to college WiFi or using the same network and mobile phone mast would have identical IP addresses. Session cookies can be set to have long expiries but a user can remove them from their browser regardless. In order to track devices with some degree of certainty, it is therefore necessary to look at alternative means of identification.

Browser fingerprinting takes various characteristics within a browser via the JavaScript API (and formally via Flash and Java virtual machines too) and uses these to essentially narrow down a browser to a very small intersection of sets. Taking a simplistic example, a browser with resolution of 1080px by 760px might have 100 possible other configurations. If we then also include the CPU class, that might narrow down the intersection to 50 devices. Then, taking the system language, it might be possible to reduce the intersection to 40 devices - and so on. On their own, these identifiers are not unique; it is only when they are combined that you reduce the probability of finding another device with an identical configuration. The Electronic Frontier Foundation run an online service [15] which will attempt to fingerprint your browser in a sandbox, displaying the results. Their test, however, is somewhat dated as another article observes [16] - relying on Flash or Java plugins to properly fingerprint the available fonts. They also note the potential for plugin detection, where the available plugins to the browser are queried and used as identifiers, given how unique the combination can be. There are some examples [17] of identifiers that might be used:

- `browserSettings.openUrlbarResultsInNewTabs`: whether URL bar autocomplete search results open in a new tab or not
- `browserSettings.homepageOverride`: the current value of the home button URL
- `runtime.PlatformOs`: the platform Operating System
- `runtime.PlatformArch` the platform architecture

These seem fairly obvious but there are more advanced techniques such as HTML5 web canvas and WebGL fingerprinting. WebGL is a graphics API that allows the programmatic drawing and rendering of images on a canvas [18]. The technique used is to draw an image onto the canvas and then convert the result back into text which can be compared against other results to determine uniqueness. In this particular study, the large changes seen in a controlled environment and relatively large execution time, led to the conclusion that WebGL fingerprinting contained too much entropy to be useful as an additional identifier to measure. The method was not described in detail however a novel approach was presented in an IEEE article which utilized both HTML5 and WebGL [19]. Key to their approach, was the rendering of text which they matched against known samples (using automated methods that are not relevant here). This was done with both WebFont (HTML5) and standard font face rendering. The WebGL test used a black and white image containing 200 polygons (derived from the ISO 12233 standard) and the various rendered images were then subtracted from a control to give a "diff" between the generated outputs for each browser, which could then be compared. The article contains some illustrations to demonstrate the point.

The actual result from a browser fingerprint is a hash of all of the attributes selected to identify the browser. This hash can then be stored, to be checked against future computations of the hash. As another paper notes in research into reverse engineering fingerprinting solutions [20], these hashes are increasingly calculated server side with the raw values extracted sent to the server. Not only is this not ideal as there is a significant transfer of quite sensitive data, but the calculations are carried out in a black box that cannot be analysed. Another variant mentioned is commercially available scripts that are loaded from a third party, returning the result directly to them for analysis and storage.[20] An example here is provided by Adyen [21] who provide customers with a script used to calculated a browser fingerprint for the purposes of their own internal fraud prevention and 3DSecure Version 2.

Proof of Concept

There are two commonly used and freely available libraries for calculating browser fingerprints - Fingerprint2 [22] and ClientJS [23], the latter of which implements some of Fingerprint2.

In order to test both libraries, I designed a simple HTML webpage with some basic HTML elements to visualize the fingerprint outputs. I subsequently wrote, according to the respective manuals, some basic fingerprinting for both Fingerprint2 and ClientJS.

It is important to note the timeout is set to allow a certain amount of time for the page to load, as partially loaded pages generate inconsistent fingerprints.

In Table 2 you see two sets of results, the output of both libraries before and after the page is refreshed. Note that only the ClientJS fingerprint remains the same. The exact reasons for this will be down to the specific parameters each chooses to use in fingerprinting. There are many options - it would take a long time to fine tune these algorithms!

I also compared the outputs in an incognito window and in Table 3 again the Fingerprint2 results do not match, whilst the ClientJS fingerprint has correctly identified the browser - despite it being in incognito mode and therefore with no cookies and no tracking.

Finally in Table 4, I compared the results across two browsers - Chrome and Firefox. Both Fingerprint2 and ClientJS identify them as different browsers, which is as be expected.

Table 2: The first set of results.

	Original	On refresh
Fingerprint2	e0cd17cbea1bdf2b801011890436d87	49a352086af089a875fd8b2a020688e7
ClientJS	4217023516	4217023516

Table 3: Firefox in normal and incognito mode.

	Normal	Incognito
Fingerprint2	6592ccb57fa24e00f32670731bda468f	1550e1931a190cb8e35c77a244d88e9f
ClientJS	2526809009	2526809009

Table 4: Chrome vs Firefox

	Google Chrome	Firefox
Fingerprint2	5e78a51f933e45e5535460a40151db9d	49a352086af089a875fd8b2a020688e7
ClientJS	3030096853	4217023516

Review

It is clear that Browser Fingerprinting is a controversial technology, with deeper ethical issues explored in the Professional Issues section. However, the Proof of Concepts have shown that it is possible to easily implement the technology but with varying results. From the brief tests above **ClientJS** has proved more reliable across different scenarios and will therefore be used further in the development of the MVP solution.

Bluetooth

Overview

In this section methods of authenticating students are explored. Mifare technology and its weaknesses are discussed, and Web Bluetooth and Web USB proof of concepts are attempted. A detailed discussion of the Bluetooth Low Energy technology and its security model is included.

Introduction

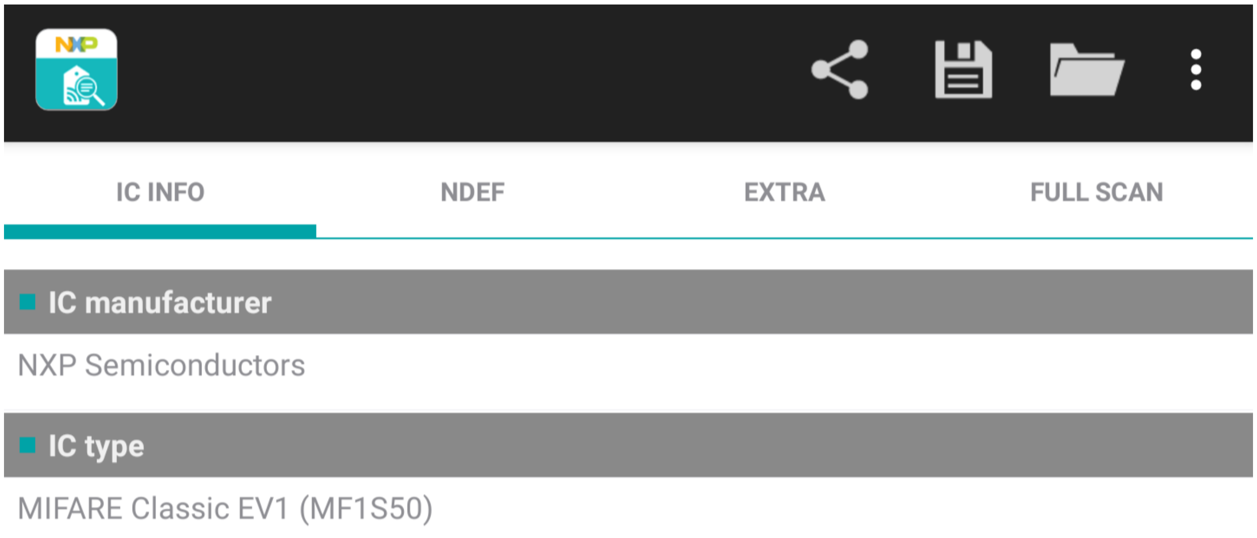
One of the key advantages that the clicker system had over the registers was that it utilised wireless communication such that students could ‘click in’ on a short range handheld device, rather than pass around a physical register sheet. The range of the clickers is advertised as being around a 200ft radius [24] which in the context of a lecture theatre, is actually quite large! Based on practical experience (and given a definite answer would be out of scope), I postulate that for most lecture theatres that would include the entire room plus a short distance outside - taking into account the attenuation of the signal through the walls.

Considering the range of other wireless communication technologies:

- Radio Frequency Identification (RFID): a maximum of 20ft [25] with a passive tag.
- Near Field Communication (NFC): a few centimeters [26]
- Bluetooth: “short range” [27]

Student ID Cards

At the beginning of the 2019 Winter Term, the College began issuing new student ID cards and reissuing older cards. The new cards were found to contain a MIFARE Classic chip (Figure 5). The MIFARE chip was


IC INFO NDEF EXTRA FULL SCAN

IC manufacturer

NXP Semiconductors

IC type

MIFARE Classic EV1 (MF1S50)

Figure 5: Screenshot of the tag read from the NXP "TagInfo" Android application

originally released in 1994 by what became NXP [28] as a product primarily for mass transit cards, but also later for door key cards and the like. The MIFARE design was very proprietary with only the (48 bit) key length disclosed to the security community. For example, the chip used an undisclosed encryption algorithm developed by Phillips (who became NXP) that had not been researched or investigated by the wider security community [28]. As is well accepted in Kerckhoff's Principle [29], we should always assume the attacker has the maximum knowledge of the system including details of the encryption algorithms used, nonce generation algorithms and other design features. The only item assumed not to be known by default are any private keys generated. This principle ensures that any tendency to default to "security by obscurity" is avoided as, as in the case of MIFARE, if your security model relies on design details being disclosed it is very vulnerable to espionage, reverse engineering or simply errors in parties disclosing too many details.

In December 2007 at the Chaos Computer Conference [30] [31], a presentation was given where researchers showed analysis demonstrating it was possible to generate the nonces used in the chip (as the randomness is determined by the number of clock cycles since the chip is initialized). By analyzing 27 mutual authentication captures repetition of the challenge response was observed. It was also shown through trial by error in bit flipping the unique identifier transmitted by the card was linked to the keys used, and such there exists for each session key as a result of the mutual authentication, for a given key and unique identifier. This allows recovery of the key used based on the unique identifier presented by the chip and the challenge response sent.

This has been developed further and it has been possible to generate key rainbow tables (as they only have a 48 bit length) and use these tables to carry out brute force searches on encrypted sectors for arbitrary cards. Note that in the dump in Appendix 3, sector 6 is unreadable - this is the secured sector for which a key is needed for the reader to access the sector. Using software implementing the later nested attack on MIFARE [32] and an Android application to interrogate the card, [33] I found that it is possible to recover this sector:

```
30313836343536342020202020202020
00000000000000000000000000000000
00000000000000000000000000000000
------------7877880045FA49A0C327
```

Where, `45FA49A0C327` is the key, `7877880045` the Access Control bits and the hexadecimal beginning `303138...` the stored data. Converted from hexadecimal to ASCII it reads `01864564`, which is my student ID number (as printed on the front of the card). With the key it is possible to rewrite any student ID card (or any MIFARE Classic card for that matter) with any ID number, or arbitrary data. Through a separate vulnerability, it is possible to retrieve any card ID number from the College LDAP extended attributes. So, it is possible to write any of these cards (used for access control, payments etc.) with any ID number and they offer no security above that of magnetic stripe technology in use on the older cards.

I therefore propose not to use the student ID cards for authentication, as they are demonstrably insecure and vulnerable to attack with little knowledge of the MIFARE technology or cryptography.

Bluetooth Low Energy

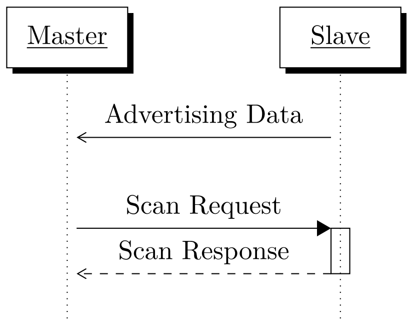
In the 4th Edition of the Bluetooth Specification [34], issued by the Bluetooth Special Interest Group, the Bluetooth Low Energy devices were introduced. These devices were to in addition to what is known as Bluetooth Classic - the most well known version of Bluetooth used in all manner of consumer accessories from headphones to keyboards, and now adopted in scientific and industrial applications. Three classes of devices were available: Bluetooth Classic (only), Bluetooth Low Energy and Bluetooth Smart Ready; the last supporting both Classic and Low Energy protocols. The intention behind Low Energy was to go almost in the opposite direction to Classic - a very low bandwidth device which would not remain constantly connected designed around applications that issued sporadic commands to retrieve data or initiate control sequences. In comparison to Bluetooth Classic at the time, Bluetooth Low Energy has a theoretical bandwidth of 1MB/s vs 54 MB/s with Classic [27] [35]. Low Energy operates on the 2.4GHz frequency band with a channel spacing of 2MHz (40 channels) with 3 channels dedicated for advertisement broadcasts - channels 37, 38 and 39 [36]. In order to minimize interference with other Bluetooth devices and other standards on the same frequency (WiFi, Bluetooth Classic, ZigBee) frequency hopping is used where the channel is changed upon each interaction based on the following formula [36]:

```
channel = (curr_channel + hop) mod 37
```

The `hop` is predetermined at the initial authentication with the Bluetooth device and remains constant throughout the life of the connection.

Gaussian Frequency Shift Keying (GFSK) is used as the modulation for transmitted signals. This is fairly common for devices operating on the 2.4GHz band and was filed for patent in 1998[37]. When radio signals are transmitted they are modulated (mixed) with a carrier signal at the frequency of transmission. GSFK uses a filter based on the Gaussian function. The exact reasons for doing this are beyond the scope of this report, but they relate to reducing the effects of interference during transmission.

The Bluetooth 4.0 specification defined two actors: a master and a slave. During the asymmetric connection process, the slave advertises itself on the advertising channels with a 31 byte packet. It is possible to send an additional 31 byte packet with additional data upon interrogation by a Master (sending a Scan Request packet) - this is Active Scanning and Passive Scanning is merely monitoring the data sent on the advertising channels.



In Bluetooth 4.0, the slave may be connected to one master device with master devices permitted to connect to a number of slave devices. This was updated in version 4.1 of the specification [38] to allow slaves to have multiple connections to a master at any one time, although slave to slave communication remains unsupported. Low Energy uses Time Division Multiplexing to achieve long periods with the radio off to conserve battery life; transmitting and receiving packets only at pre-determined intervals. This can give a battery life measured in years, but does mean the bandwidth is limited.

The Bluetooth stack uses L2CAP (Logical Link Control and Adaptation Protocol) at second lowest level of the stack (the Physical Layer handling the modulation and transmission of packets is below) and this is responsible for the assembling of packets and managing the Physical layer to ensure downstream units, such as the HCI, are not overwhelmed.

Bluetooth defines a number of modules including the Generic Attribute Profile and the Generic Access Profile, which help abstract services away from the hardware and provide common interfaces between devices. The exact mechanisms of these and the general design is beyond the scope of this report - we simply accept that the Bluetooth specification provides these services for our convenience.

However, the security mechanisms provided are of interest especially given the weaknesses noted in MIFARE.

Security Manager

One module is the Security Manager which handles the mutual generation of keys for encrypted communications and interfaces with the L2CAP module and is only present in Low Energy devices (in Classic devices it is integrated in the Controller). Using hardware modules, it is able to provide various cryptographic functions for the key exchanges.

The security function used is an AES-128-bit block cipher. The Advanced Encryption Standard [39] was chosen as a result of a RFP issued by the United States National Institute of Standards and Technology looking to replace the then aging and provably broken Data Encryption Standard (DES), developed many years before. The algorithm chosen was Rijndael [40] and operated on 128, 196 and 256 bit blocks and is a symmetric cipher. AES works by applying a round function to the data input using the expanded key as a state. On each round, a byte substitution, then row movement and finally column function is applied to the data. In each round, the designated part of the key is XOR'd with the state. There are 10, 12 or 14 rounds dependant on key length and the last round does not mix the columns. Applying the algorithm in reverse is possible and allows decryption of the ciphertext.

When Bluetooth devices are paired, a STK (Short Term Key) is generated to encrypt data symetrically with AES in transit. Pairing is carried out via three methods:

- Just Works: values to generate STKs are exchanged with no encryption over the link offering no man-in-the-middle protection. Interception of the values allows an attacker to generate their own copy of the STK and decrypt and potentially spoof further communications.
- Passkey: a 6 digit numeric code is assigned to the Slave device and on pairing the Master requires user input of this code which is then used to generate a key to encrypt exchanged values for the STK generation.
- Out of band: values are exchanged outside of the Bluetooth protocol to allow encryption of the values to generate the short term key.

It should be noted that in the proposed implementation, Just Works pairing will be used. It is possible to use Passkey pairing, however, since each student would know the identical code it would offer no protection against them intercepting their own packets and thus has negligible security benefits as they are the main adversaries. There are also technical limitations to dynamically changing the Passkey on Bluetooth Low Energy Slaves and this is currently not properly supported.

BLE Device Investigations

Investigations into practical applications and small projects with Bluetooth Low Energy yielded several articles mentioning the HM-10 [41]. The HM-10 is based on a Texas Instruments BLE System On Chip mounted as a daughter board to breakout a serial UART connection. At this time, only BLE devices are officially supported via the Web Bluetooth APIs [42].

As this device provides a serial interface, an additional microprocessor is required to interface with the device and to receive, process and then respond to data sent to the device. A Raspbery Pi was chosen over an

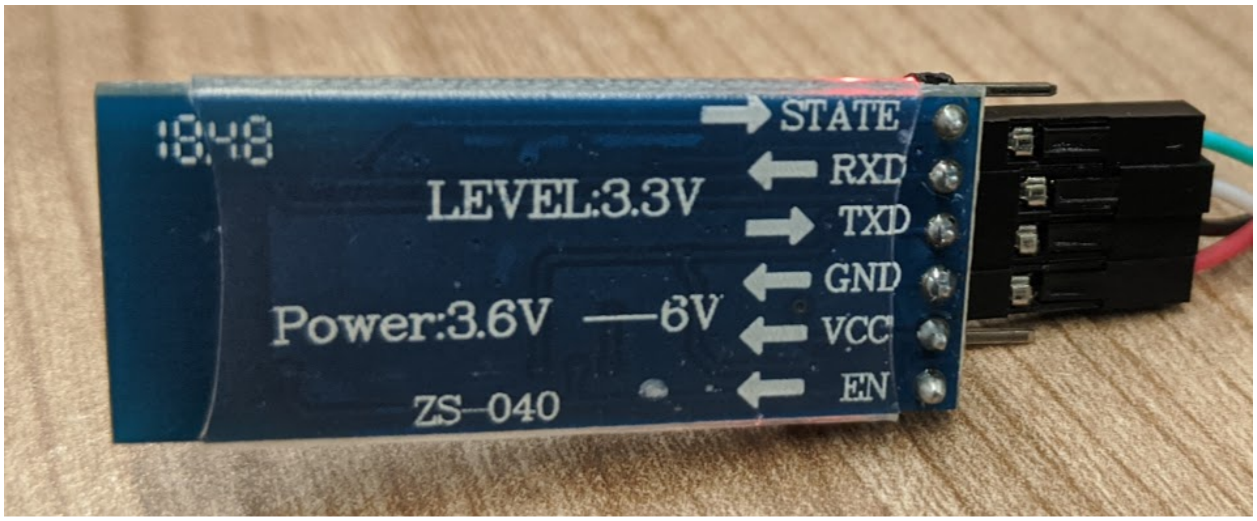


Figure 6: A HM-10 module

Arduino due to the increased processing power, onboard networking and full operating system. This also allows for higher level languages like Python to be used instead of C++, speeding up development. As no analogue inputs are required, there is no real advantage to using an Arduino here. A Rasperry Pi Zero was therefore configured to run in headless mode with a terminal over USB [43]. However, it transpired that whilst the Raspberry Pi does have two UARTs available, one is used for the onboard Bluetooth and the UART used for the Linux console is shared with the UART exposed on the GPIO [44]. It was therefore necessary to change approach and a Raspberry Pi 2B was provisioned with SSH enabled (add an empty file: `/boot/ssh`) such that a direct ethernet connection to the laptop was possible, with ethernet passthrough to the laptop's wireless connection.

The HM-10 was then attached with Dupont leads to the GPIO of the Raspberry Pi using pinouts for the Pi and the markings on the HM-10 breakout board. [45]

Initially, the device was detected on the serial port and a connection could be established. However, there was no data sent by the device and no acknowledgement of data sent down the serial port. The HM-10 uses AT commands which are commonly found in GSM modules and the like - the HM-10 likely uses these given the industry crossover. The manual for the HM-10 [46] lists a few basic AT commands which can be used to interrogate basic data such as the device status:

- `AT` should return the device status: `OK` or `OK/LOST`
- `AT+ADDR?` should return the device MAC: `OK+ADDR:{MAC address}`

The datasheet also describes that a string over 80 characters should be sent to take the device out of standby. The string data is not relevant, just the length.

The manual states the serial parameters are:

- 9600 Baud
- No parity
- 1 stop bit
- 8 bit byte length

A few attempts were made using these settings and different serial decoders (Minicom, Screen) but no response was received. Appendix 4 contains a sample output from Minicom for one attempt (including configuration).

It transpired that AT commands are actually a time based command set. They do not rely on a carriage return or line feed to indicate the end of a command, but they do timeout - within 100ms usually [47]. A timeout on a valid command will cause the command to be executed and the result returned. An invalid

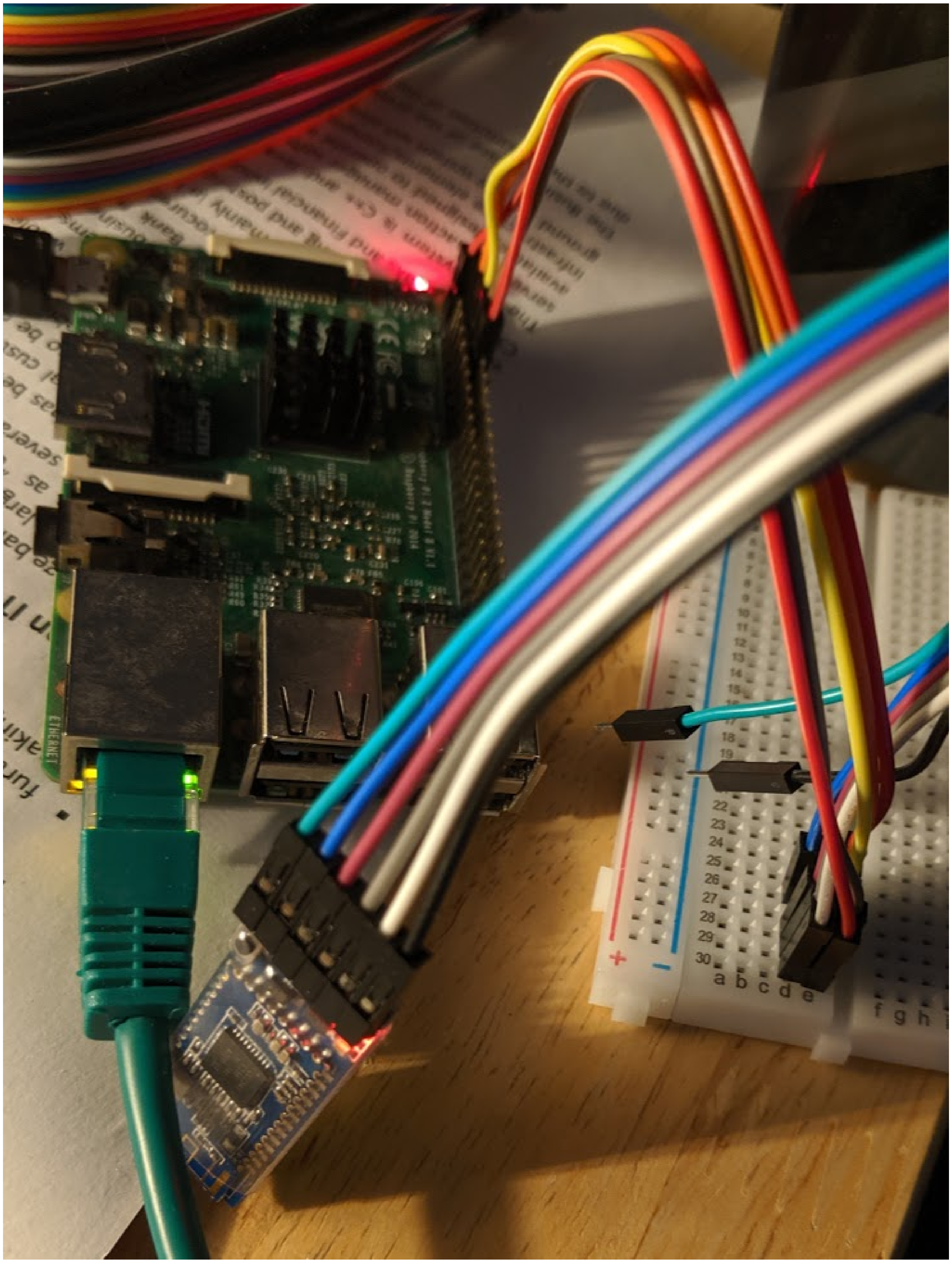


Figure 7: Raspberry Pi 2B with an HM-1o breakout board attached

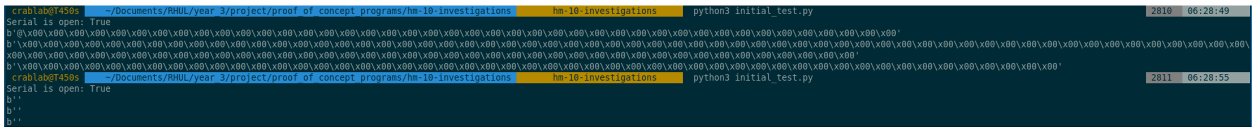
command may generate no response at all.

Looking at the Raspberry Pi forums, I identified a small number of questions relating to this, including a thread where a user had reported some working test code [48]. The code is actually very trivial - it opens a serial connection with the correct parameters and then sends data before listening. Because AT commands use a 100ms timeout it is not necessary to run two threads to send and receive data - after sending a command it is simply necessary to wait for any data on the serial buffer for 100ms before continuing. I modified the script to:

- Send the long string (above 80 characters)
- Send the `AT` command and listen for a response
- Send the `AT+ADDR?` command and listen for a response

The script is provided in Appendix 5.

Running this script several times with different RX/TX pin configurations and baud rates did not yield any results and the only success was some null bytes, as in Figure 8.



```
 crablab@T450s   ~/Documents/RHUL/year_3/project/proof_of_concept_programs/hm-10-investigations   hm-10-investigations   python3 initial_test.py                                   2810   06:28:49
Serial is open: True
b'@\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
 crablab@T450s   ~/Documents/RHUL/year_3/project/proof_of_concept_programs/hm-10-investigations   hm-10-investigations   python3 initial_test.py                                   2811   06:28:55
Serial is open: True
b''
b''
b''
```

Figure 8: Run one with null bytes, second run with nothing. The change between runs is swapping the TX/RX pins on the UART.

Reading around the UART protocol, it transpired that flow control is sometimes required for devices [49]. Flow control is used to indicate to the other party when it is 'safe' to send data and when it is not. As UART is so ubiquitous it is likely there will be cases where the buffer size and the processing capabilities between devices are vastly different. To prevent more capable devices flooding low powered devices (either leading to lost data or exceptions) the crosscoupled `state` (RTS) and `enable` (CTS) supplement the data lines and when a device needs to inhibit further data it will bring the `state` line to Logic 0 (`enable` on the partner device) [50].

The Raspberry Pi does support hardware flow control, however enabling it is non-trivial and it was not possible to enable it in Minicom, after looking into several methods. [51]

Upon speaking with Dave Cohen and investigating further, it was suggested that as the RX pin of the HM-10 was technically 3.3V, providing a 5V signal could cause issues. A potential divider was tried but this also failed to resolve the issues [52]. It also appeared that later versions of firmware used the higher 15200 baud rate - this was also tested.

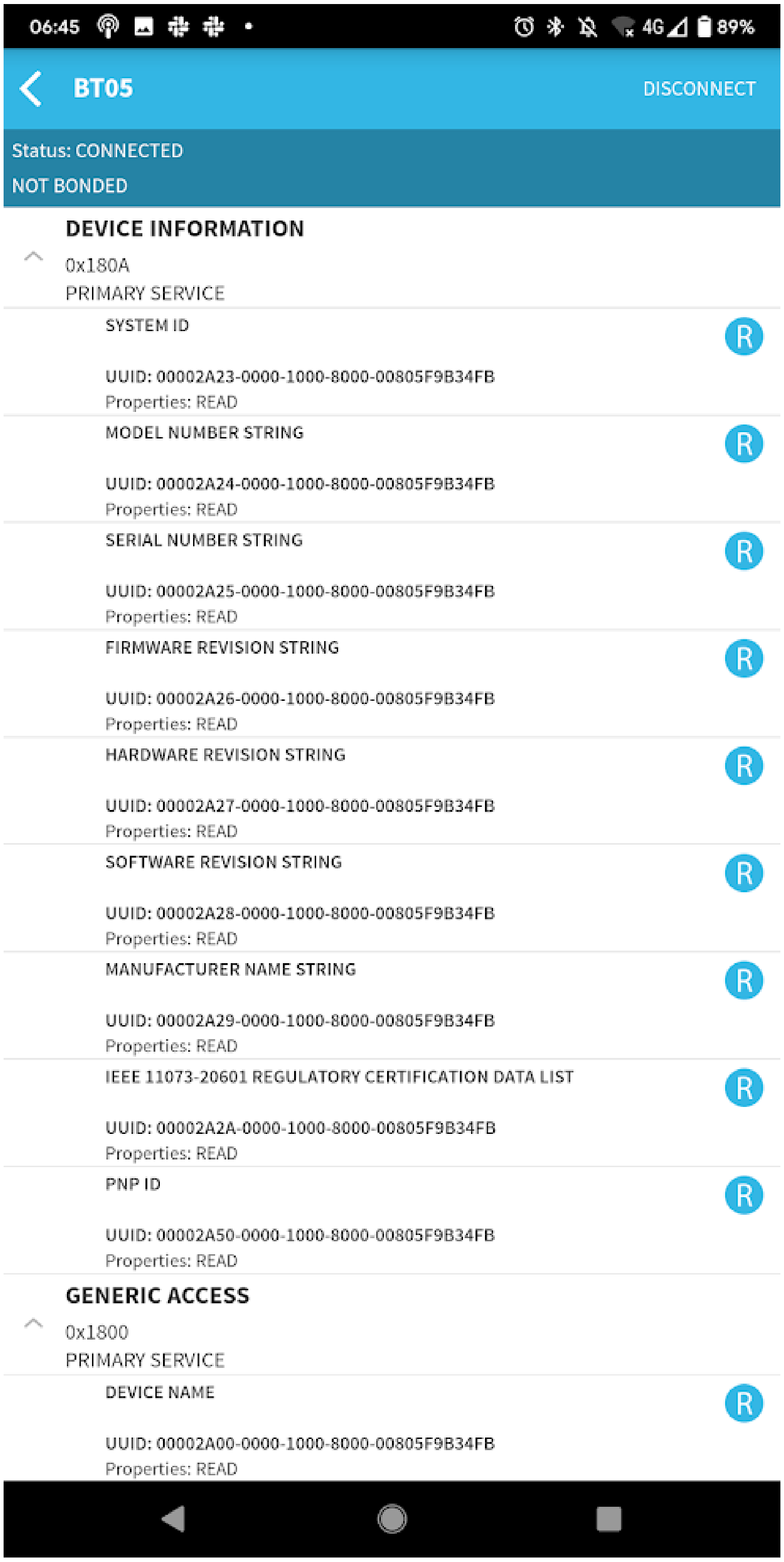
It was also observed that using a BLE Scanner Android application (Figure 9), it was possible to connect to the HM-10 and read out the GATT data - no acknowledgement of the connection was observed on the serial port as expected and documented. This suggested the device was, in part, functioning as expected.

Up until this point, two HM-10 devices from the same source had been used. Due to the issues encountered another was procured from an alternative supplier (Figure 10). Upon swapping it like for like with the other HM-10, it worked first time with the script provided in Appendix 5. This does validate that this was a hardware issue rather than "user error" however the exact problem remains unknown.

Using a BLE Terminal app, it was then possible to send data via the HM-10 to a running screen session as in Figures 12 and 13:

The support for the Web Bluetooth APIs is rather more limited than I had believed and only Chromium and Opera offer any support for the APIs [53]. The officially updated development progress can be found here.

Based on a development guide from Google (written by a member of the Web Bluetooth Community Group) [54] a very basic solution was developed (Appendix 6) and there are several items to note.


06:45 4G 89%

BT05 DISCONNECT

Status: CONNECTED
NOT BONDED

DEVICE INFORMATION
0x180A
PRIMARY SERVICE

SYSTEM ID
UUID: 00002A23-0000-1000-8000-00805F9B34FB
Properties: READ

MODEL NUMBER STRING
UUID: 00002A24-0000-1000-8000-00805F9B34FB
Properties: READ

SERIAL NUMBER STRING
UUID: 00002A25-0000-1000-8000-00805F9B34FB
Properties: READ

FIRMWARE REVISION STRING
UUID: 00002A26-0000-1000-8000-00805F9B34FB
Properties: READ

HARDWARE REVISION STRING
UUID: 00002A27-0000-1000-8000-00805F9B34FB
Properties: READ

SOFTWARE REVISION STRING
UUID: 00002A28-0000-1000-8000-00805F9B34FB
Properties: READ

MANUFACTURER NAME STRING
UUID: 00002A29-0000-1000-8000-00805F9B34FB
Properties: READ

IEEE 11073-20601 REGULATORY CERTIFICATION DATA LIST
UUID: 00002A2A-0000-1000-8000-00805F9B34FB
Properties: READ

PNP ID
UUID: 00002A50-0000-1000-8000-00805F9B34FB
Properties: READ

GENERIC ACCESS
0x1800
PRIMARY SERVICE

DEVICE NAME
UUID: 00002A00-0000-1000-8000-00805F9B34FB
Properties: READ

Figure 9: Using the BLE Scanner Android application to connect to the HM-10

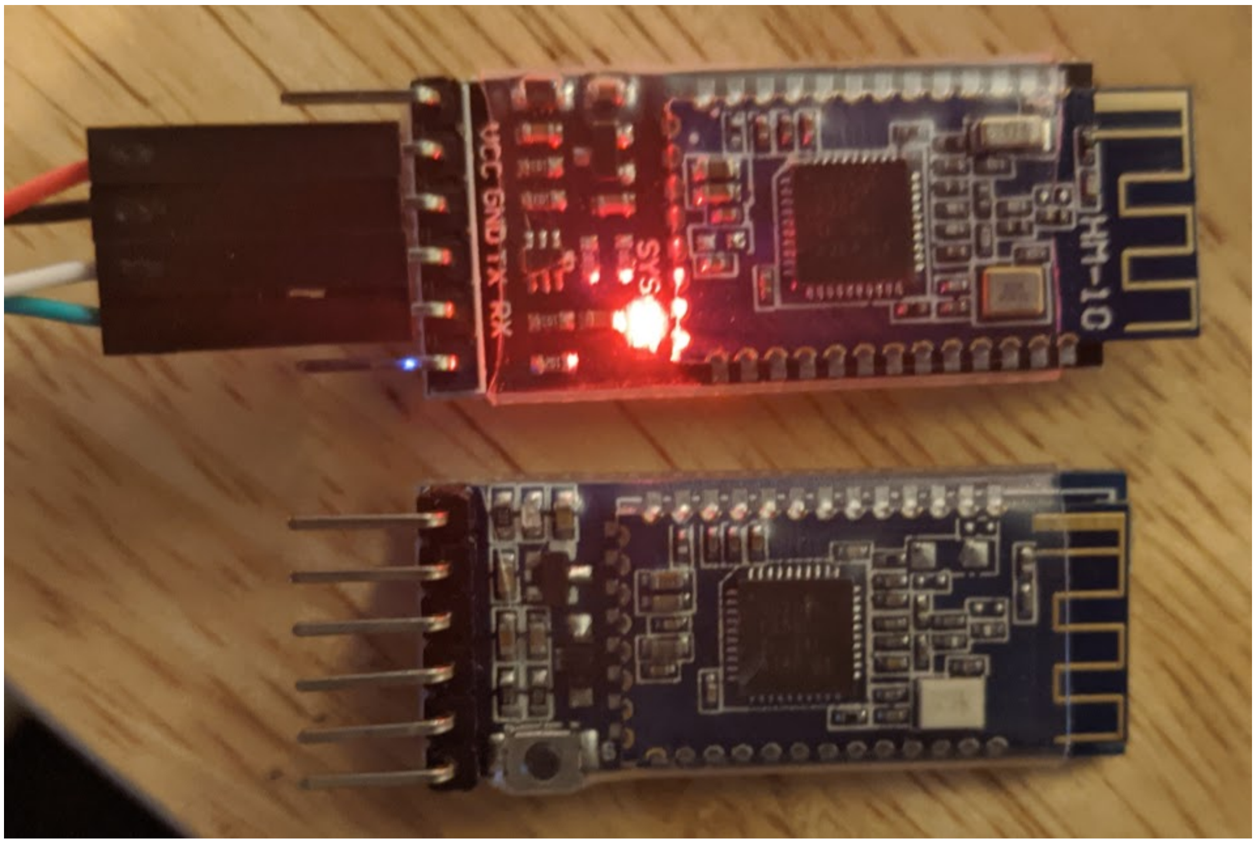
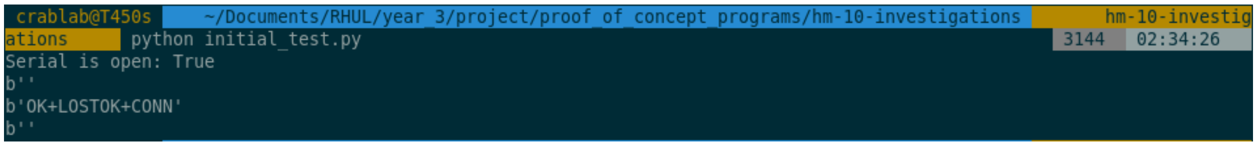
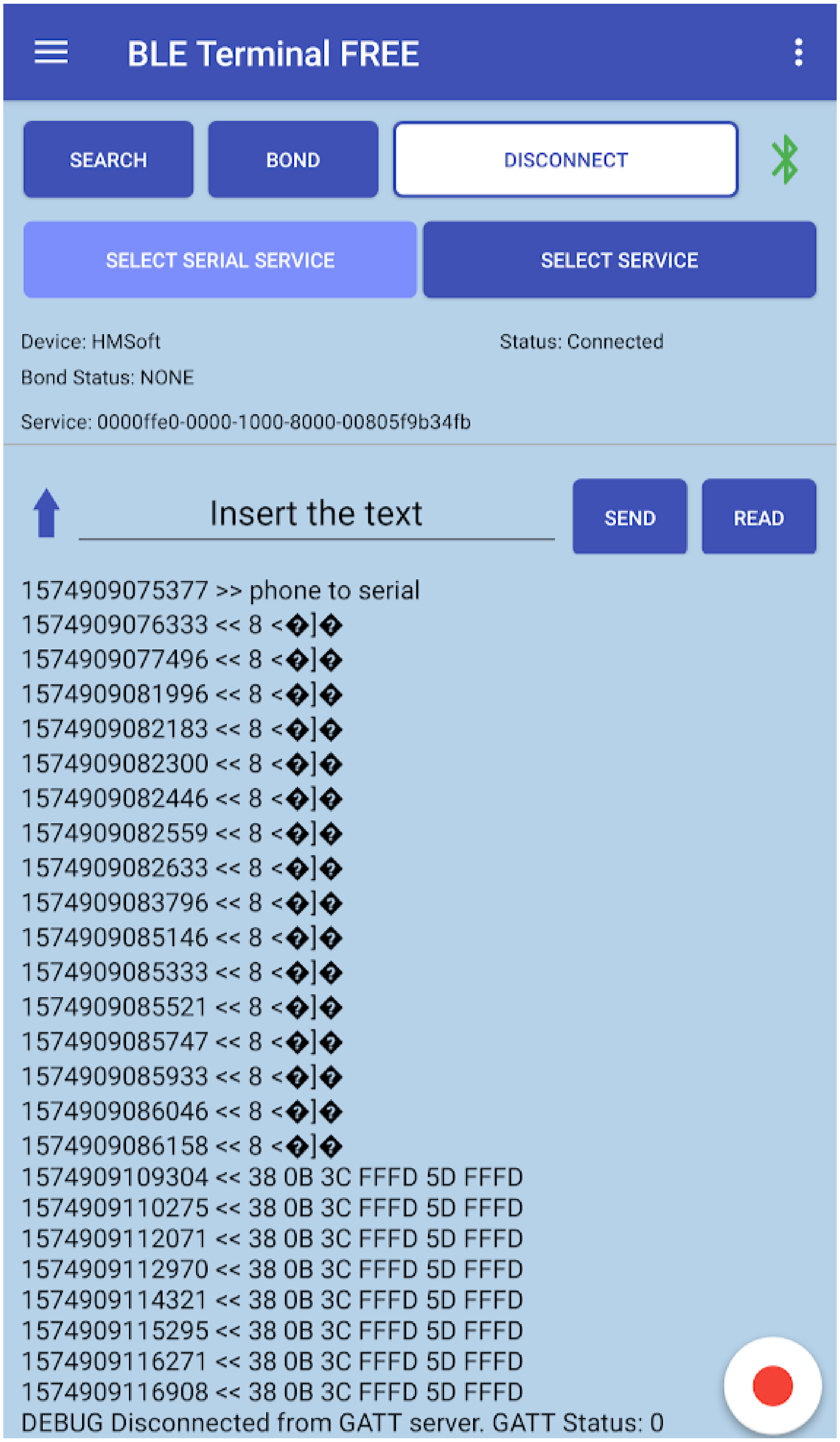


Figure 10: The two HM-10s side by side. The new (and working) device has the LED illuminated



```
 crablab@T450s    ~/Documents/RHUL/year_3/project/proof_of_concept_programs/hm-10-investigations       hm-10-investig
ations       python initial_test.py                                                                   3144   02:34:26
Serial is open: True
b''
b'OK+LOSTOK+CONN'
b''
```

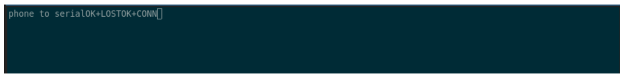
Figure 11: The expected output of the script in Appendix 5 with a working HM-10



Device: HMSoft
Status: Connected
Bond Status: NONE
Service: 0000ffe0-0000-1000-8000-00805f9b34fb

```
1574909075377 >> phone to serial
1574909076333 << 8 <�]�
1574909077496 << 8 <�]�
1574909081996 << 8 <�]�
1574909082183 << 8 <�]�
1574909082300 << 8 <�]�
1574909082446 << 8 <�]�
1574909082559 << 8 <�]�
1574909082633 << 8 <�]�
1574909083796 << 8 <�]�
1574909085146 << 8 <�]�
1574909085333 << 8 <�]�
1574909085521 << 8 <�]�
1574909085747 << 8 <�]�
1574909085933 << 8 <�]�
1574909086046 << 8 <�]�
1574909086158 << 8 <�]�
1574909109304 << 38 0B 3C FFFD 5D FFFD
1574909110275 << 38 0B 3C FFFD 5D FFFD
1574909112071 << 38 0B 3C FFFD 5D FFFD
1574909112970 << 38 0B 3C FFFD 5D FFFD
1574909114321 << 38 0B 3C FFFD 5D FFFD
1574909115295 << 38 0B 3C FFFD 5D FFFD
1574909116271 << 38 0B 3C FFFD 5D FFFD
1574909116908 << 38 0B 3C FFFD 5D FFFD
DEBUG Disconnected from GATT server. GATT Status: 0
```

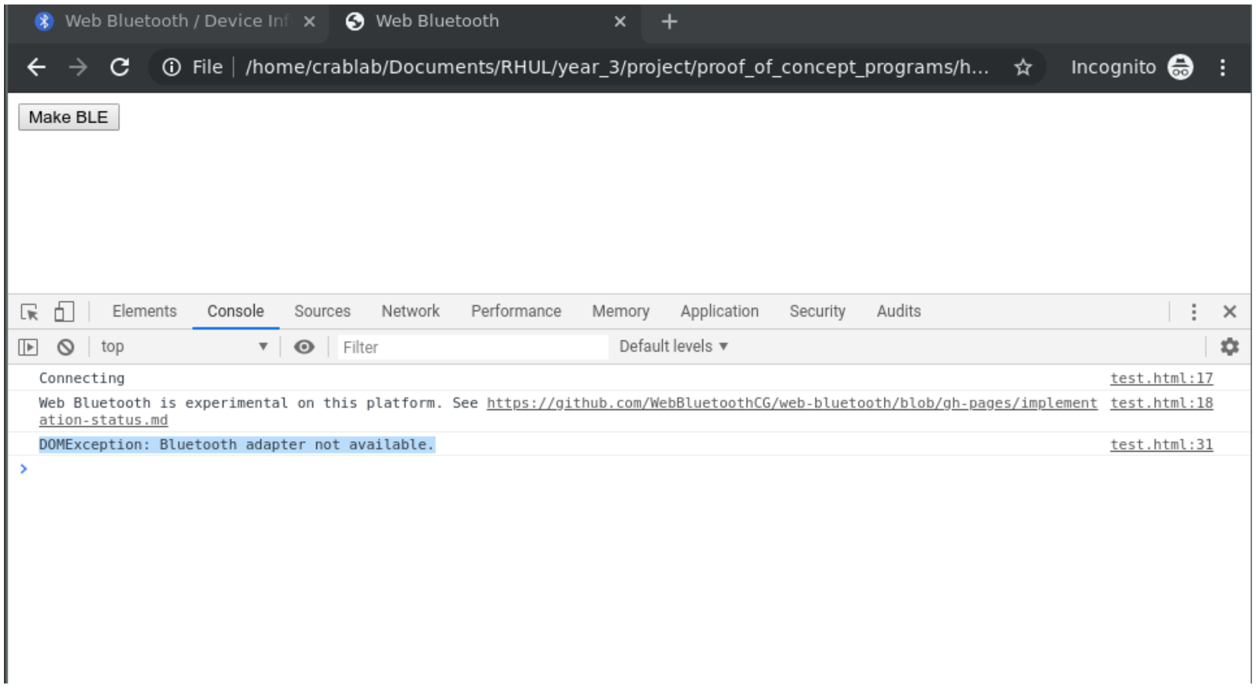
Figure 12: The BLE Terminal Android application connected to the HM-10 and transmitting and receiving text.



```
phone to serialOK+LOSTOK+CONN
```

Figure 13: The equivalent end with a screen session attached, showing the text and the AT messages when GATT changes take place.

The Web Bluetooth API requires user interaction on the page before scanning can even take place, and in any event the user needs to explicitly provide consent to pair with a device via a prompt. In the solution, an `onClick` event from a button is used to call the function. Once called, the function scans devices and looks for an explicit unique identifier for the specific HM-10 in use (transcribed from the BLE Scanner app): `0x484D536F6674`. It is possible to apply filters based on GATT characteristics, however, for simplicity this is avoided. The name of the device should then be logged, and the pairing sequence initiated (requiring user confirmation).



```
Connecting                                                                                test.html:17
Web Bluetooth is experimental on this platform. See https://github.com/WebBluetoothCG/web-bluetooth/blob/gh-pages/implement  test.html:18
ation-status.md
DOMException: Bluetooth adapter not available.                                            test.html:31
```

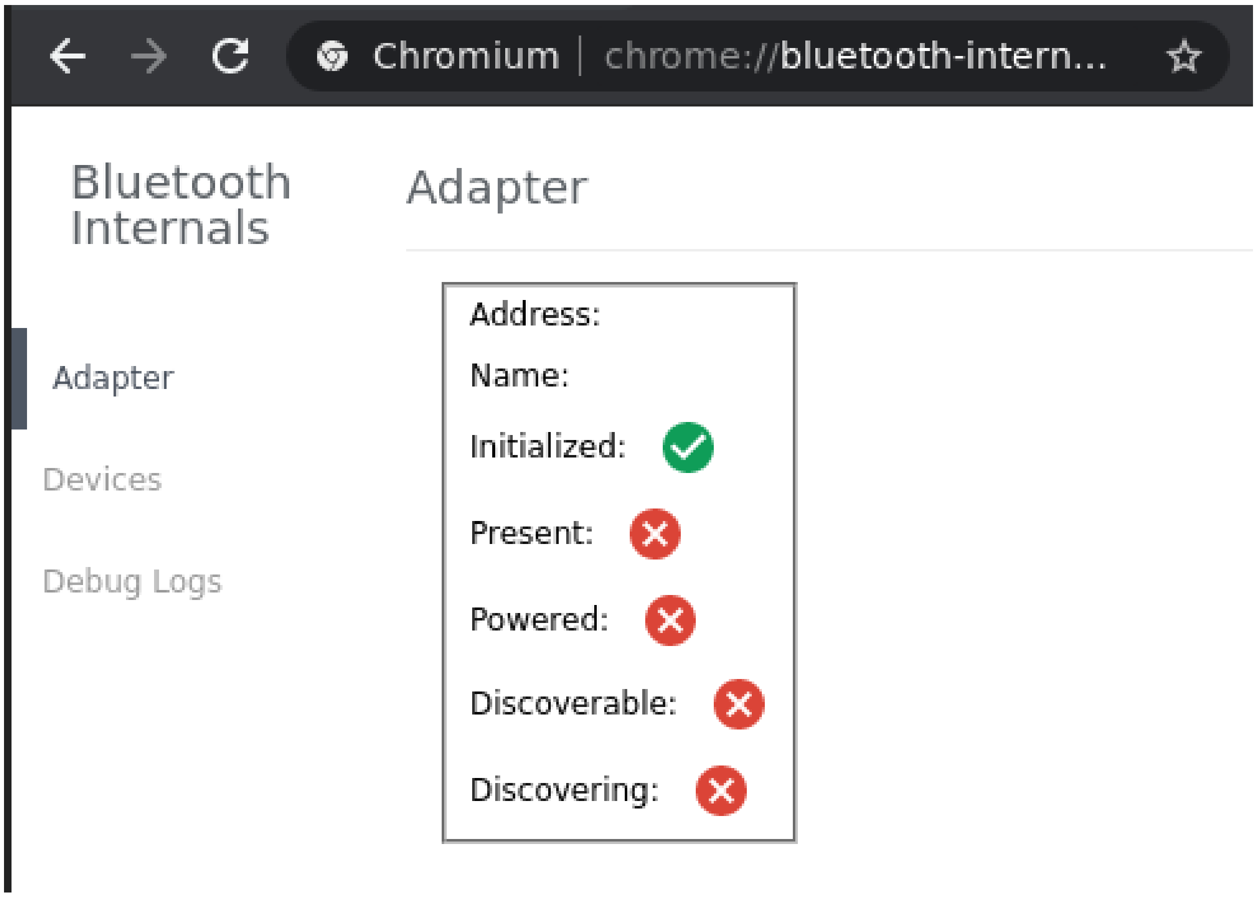
Figure 14: Web Bluetooth API issues with Linux Bluetooth drivers

Web Bluetooth is so experimental that it requires enabling a Chromium feature flag [55] and on Linux, enabling an experimental flag for Bluez [56]. As can be seen in Figure 14 and Figure 15, despite using a compatible Bluetooth adaptor recognised by Bluez, it was not possible get the Web Bluetooth API working on the hardware available to me.

As an alternative for the proof of concept, I looked into the Web NFC library. This allows you to read/write to NDEF tags from the Chrome browser. Low level protocols are not exposed so to "exchange" data in a challenge reponse a workaround would be required. For example, writing to the "tag" and then reading it straight after with the microcontroller responding with a signed version of the previously written data instead. The Web NFC library is however similarly unstable to the Web Bluetooth library and is not widely implemented beyond some limited alpha testing which was announced in March [57].

Instead, I planned to use the Web USB library to demonstrate the concept of students registering using a crytographic signature from a seperate device [58].

Web USB works with a more limited set of Arduino devices due to the way different Arduinos provide control over the USB interface [59]. I elected to use an Arduino Micro, one of the listed devices known to work with



Address:

Name:

Initialized: ✓

Present: ✗

Powered: ✗

Discoverable: ✗

Discovering: ✗

Figure 15: Web Bluetooth Chrome Debugging tools showing the driver/adaptor issue

the specification. A basic sketch is provided using the Web USB Arduino library and an example console application at: https://webusb.github.io/arduino/demos/console/

In Linux you also need to add a `udev` rule to allow the browser to communicate with the Web USB device and add your user to the `plugdev` group.

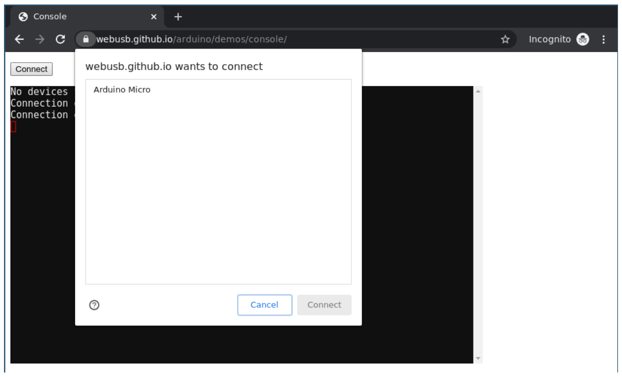
I ended up using a more specific version of the rule provided in the documentation [58] as the rule in the official documentation did not appear to work:

```
SUBSYSTEM=="usb", ATTR{idVendor}=="2341", ATTR{idProduct}=="8037", MODE="0666", GROUP="plugdev"
```

Confirming my user is in the `plugdev` group:

```
$ cat /etc/group | grep "plugdev"
plugdev:x:46:crablab
```

This allowed the Web USB device (the Arduino Micro) to be recognised by the browser (Figure 16), however when trying to "connect" Chrome threw a security exception (Figure 17). Despite a lot of troubleshooting, modification of the `udev` rule and searching, I was unable to get this to work.


Console
webusb.github.io/arduino/demos/console/
Incognito
Connect
webusb.github.io wants to connect
Arduino Micro
Cancel
Connect

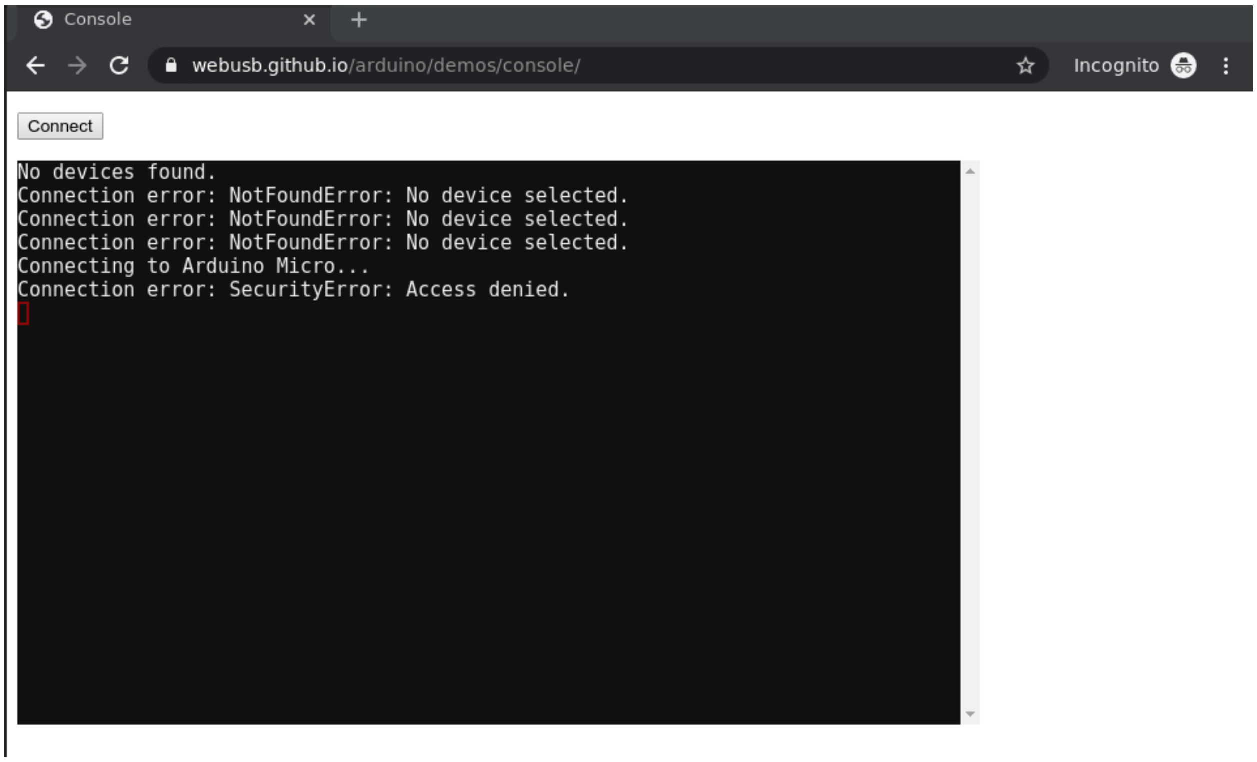
```
No devices
Connection
Connection
```

Figure 16: Web USB detecting and prompting connection to the Arduino Micro

It transpires that it is still reasonably difficult to connect to physical, external devices through a web browser. In the age of the Internet of Things this is perhaps surprising, but it is clear that further research into the wide range of APIs available (and how usuable they are in practice) is needed. For this project it is a real shame that it has not been possible to find a working solution in the time available, and this is explored further in the Self Evaluation sections.

Conclusion

Over the course of my research I have investigated a host of new technologies, as well as conducting an analysis the security of the existing clicker system. It was concluded that the current clicker system is vulnrable to both simple replay attacks and more involved impersonation attacks. Research into alternative technology looked at Mifare, Bluetooth security and standards as well as the Web Bluetooth API and Web USB API. Although it was not possible to ultimately get either technology working in the time available, it did demonstrate the fragility of these technologies at the moment, as well as the potential should more work be invested to improve stability. Additional research to design countermeasures to attack of the proposed system were also explored in the use of browser fingerprinting technologies, with proof of concepts successfully demonstrated. Ethical issues were considered, and have been discussed more widely in the professional issues section.



```
No devices found.
Connection error: NotFoundError: No device selected.
Connection error: NotFoundError: No device selected.
Connection error: NotFoundError: No device selected.
Connecting to Arduino Micro...
Connection error: SecurityError: Access denied.
```

Figure 17: The Chrome exception when connecting to the Arduino Micro Web USB device

Design and Software Engineering

Overview

The main software deliverable is an application to demonstrate a proof of concept registration system using technologies previously explored and researched in the report. This section goes through the design process and then provides insight into the build process, and the issues encountered.

Problem Statement

Royal Holloway keeps track of attendance in lectures both to ensure that students are regularly attending and also to satisfy legal requirements regarding the visas of overseas students [1]. It is essential that this data is gathered and analysed efficiently and accurately.

In the past attendance has been tracked using signatures on registers. More recently, due to the lack of scalability of this former approach, a system of clickers has been employed [2]. This latter system is non-optimal and insecurities have been discussed in another report.

In the context of a lecture there are several key problems faced:

- **Circulation of a register**: In classes that range from 50 to 200 students it is infeasible to expect a paper register to have made a complete circuit of the room by the end of a class. Adding another register doesn't always solve this, as the paths of the two registers will often intersect and students are less interested in the optimal pathing of the register than the lecture content. There is often a "scrum" at the end of a lecture between those who did not sign to register their attendance and who need to leave in time for their next class.
- **Data processing**: When discussing paper registers, it is obvious that significant administrative effort is required to process the written records and translate them to a digital medium for easy statistical analysis. This becomes more complex when there are multiple registers for a class, or when changes

to the document in question are made. For instance, sometimes students have not been included on the sheet due to an oversight and they will then append their name to the end of the document to register their attendance in a non-standard fashion. Whilst with the digital clickers there is no need more manual transcription, the data from the Turning Point software is in a proprietary format and requires conversion to a CSV and then further processing to be handled by the custom software used in the department for attendance monitoring.

- **Forgery**: Both the clicker and paper register systems are vulnerable to attack. The paper register can simply be signed by another person on behalf of an absentee. The way to prevent this is to carry out signature checking against a sample on record and random ID checks at lectures where forgery is suspected. The signature comparison is labour intensive and only catches bad forgeries. Student signatures also change over time and can be "gamed" to provide trivial samples which are easily forged. The attack on the clicker system has been described earlier in the Research section.
- **Intrusiveness**: The paper register is relatively intrusive in a lecture, however more so to the students than to the lecturer. The clicker system, anecdotally, tends to cause more issues as the set up process is non-trivial cutting into lecture time and lecturer patience. There is no real feedback for a student that their attendance has indeed been counted in the same way as there is on a paper register. The clicker system also requires you to remember a device that must be in your possession to verify your identity - a device is easy to forget, a signature less so.

This section lays out the design of the core components of the proposed system and is intended to provide a full specification together with justification for the various design choices.

Functional Requirements

These function requirements are based on User Stories, included in Appendix 8.

1. Student
    a) A website for students to access to manage their attendance
    b) An average percentage attendance score for students for their currently ongoing classes
    c) An individual percentage attendance score for students for a given class
    d) Data exports of attendance by class
    e) Capability to initiate registration process for a class to mark a student as having attended
        1. Website to initiate wireless connection with device in lecture theatre
        2. Website to send data to be signed by wireless device, and receive response
        3. Website to record correctly signed responses as attendance
    f) Ability to make absence request for a given time period
        1. Ability to view absence requests and their status
        2. Ability to withdraw an absence request
2. Lecturer
    a) Lecturers to be able to view their classes on a website
    b) Lecturers to be able to see overall percentage attendance for their class
    c) Lecturers to be able to see breakdown of percentage attendance by student per class
    d) Lecturers to be able to see breakdown attendance records by student per class
    e) Lecturers to be able to provision wireless device for an upcoming class
        1. This involves providing a configuration file and certificate to the device
3. Administrator
    a) Administrators to be able to view classes by department on a website
    b) Administrators to be able to view same breakdown of data as lecturers
    c) Administrators to be able to view student overall attendance across all classes

d) Administrators to be able to view students whose attendance falls below required minimum
e) Administrators to be able to view students whose days of absence is above a required threshold
f) Administrators to be able to manage absence requests
    1. Ability to review outstanding requests
    2. Ability to approve or deny requests
    3. Ability to view previous requests per class or per student
g) Administrators to be able to view browser fingerprinting data
    1. Ability to see when a fingerprint is used across multiple students
    2. Ability to see fingerprint consistency per student
h) Arbitrary data export as CSV for all previous views
    1. This in particular concerns a format the College can process - student ID against a boolean attendance value for each class in a CSV.

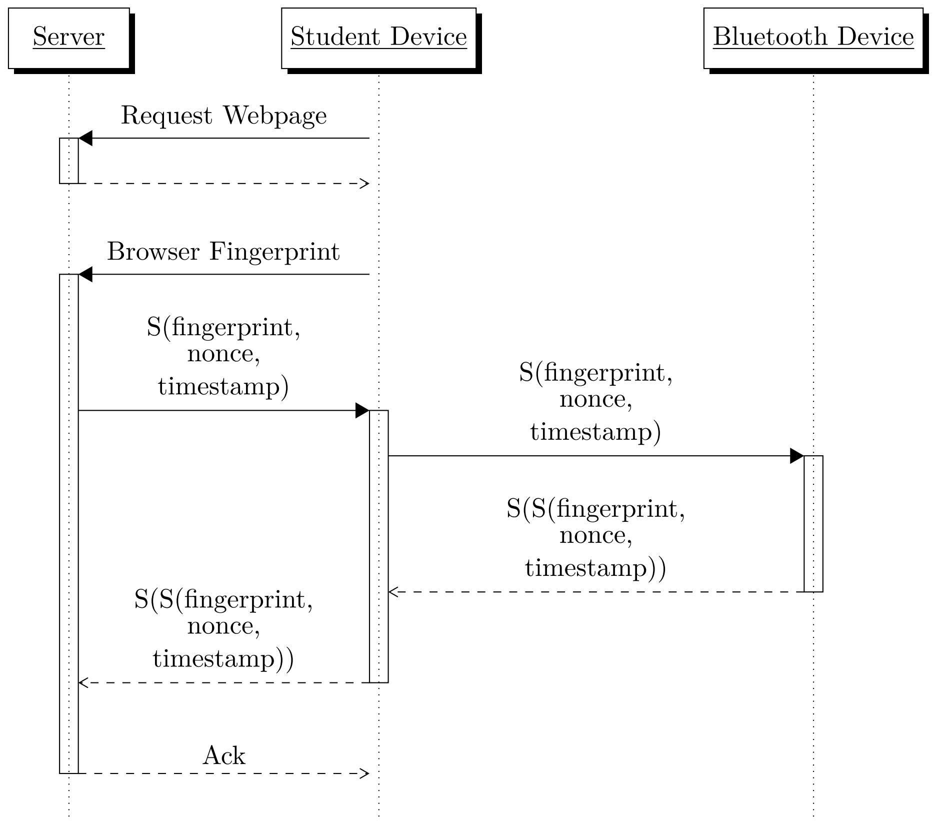
Initial Design

The proposed proof of concept replacement for both the paper registers will make use of modern web technologies to deliver a secure and trustworthy registration system that students can use on their mobile devices. This will involve a Bluetooth Low Energy device that a lecturer will configure via a website prior to the lecture, and will be plugged in to a power source for the duration of the lecture in the lecture theatre. No networking or continous connection to a PC will be required for this device. Students will log in to a website and request to register for the lecture. The website will, using the Web Bluetooth API, connect to the Bluetooth device in the lecture and will request that the device signs some data previously signed and provided by the server, as a challenge. This will provide proof that the student was in, at least the vicinity, of the lecture.

Using the research I have previously carried out, the system will have the following high level functionality:

- Login system for students, administrators and lecturers
- Functionality to create lectures and assign students to the lecture
- Use of an Elliptic Curve Digital Signature Algorithm (ECDSA) to implement a trusted challenge-response system
- Use of the Web Bluetooth API so a student device can connect to a lecturer device (physcially present in the lecture hall) to pass the student challenge
- A lecturer device which will support the Bluetooth Low Energy specification and accept a challenge, signing it with a pre-shared key and returning the response to the student
- Browser fingerprinting libraries to record the fingerprint of the browser the student submits their challenge with to prevent collusion
- Analytics to monitor registrations and provide appropriate statistics

For completeness, the challenge response elemement of the flow can be expressed in the following UML Sequence Diagram:


Server
Student Device
Bluetooth Device
Request Webpage
Browser Fingerprint
S(fingerprint, nonce, timestamp)
S(fingerprint, nonce, timestamp)
S(S(fingerprint, nonce, timestamp))
S(S(fingerprint, nonce, timestamp))
Ack

Analysis of the Initial Design

- There is nothing to stop someone signing in and then simply walking out; or indeed standing just inside the range of Bluetooth Light but outside the lecture theatre. This is not a risk mitigated by either the register or the clickers. It could be possible to have the student device poll for signatures throughout the lecture, however this would potentially have technical constraints around student device battery life, the capacity of the Bluetooth signing device and running background processes in a web browser. Ultimately, Bluetooth Light is not designed for these kind of constant connections so Bluetooth Classic (which is more difficult to implement) would be a more appropriate communication medium. An alternative would be to randomly poll devices, however the Web Bluetooth implementation requires the user to knowingly acknowledge a connection each time [42] and the proposed implementation is designed to accommodate that, so this potential solution is beyond the scope of this project.
- It is conceivable that students might attempt a relay attack by generating the data to be signed remotely and then passing it to another student to transmit it for signing, before they submit the signed data back to the server without ever having been near the lecture. There are two ways this could be solved:
  - Firstly, including the student Bluetooth MAC address in the data signed by the server and validating it with the source MAC address of the transmission on the Bluetooth device would allow you to ensure the device that sent the message also generated the message. This then requires the Bluetooth device to carry a certificate used by the server to validate the message it has sent - this isn't a bad idea regardless, although having the Bluetooth device blindly sign messages regardless isn't a huge security concern as a brute force attack is infeasible. Technically, Bluetooth devices are supposed to have a fixed and unique MAC address which will not change - plus a configurable MAC that does. However, device manufacturers have begun to randomize even the supposedly fixed MAC addresses for privacy reasons, so these may not be a reliable identifier. [60]
  - Secondly, implementing timing based controls to detect the round trip time of a message to ensure it does not exceed a certain value. This is the method used in EMV payment cards [61], known in the specifications as "Relay Resistance Protocol"). This is vulnerable to attack should a rogue

card reader not carry out the timing checks correctly. [62] Our security model here is different, as we make the assumption that both the server and the Bluetooth device are not compromised and as any modification of the data would be detected (as the signatures would no longer match) it would not be possible for the student device to alter the timestamp to carry out such an attack. This is the solution that will be implemented, subject to time constraints.

- The proposed implementation will use simple username/password authentication and if any user/password management is provided it will be basic. This is not within the scope of the project and it is likely that if this were ever to be used it would be necessary to integrate this with existing user accounts; such as via the LDAP.[2]

Components

Webservices

The proposed application is primarily a webservice which is made up of two components: a backend server for processing and storing data, and a set of frontend webpages which will allow users to interact and manage the application.

There is a choice between whether to use backend driven views or one of many JavaScript web frameworks (such as Vue, React etc.); which provide a more interactive and arguably seamless experience. These frameworks use static HTML pages served by a CDN (Content Delivery Network) which then utilise APIs provided on the backend to populate the pages on demand and deliver a richer user experience. This method of API centric design also reduces coupling of the user views with the underlying classes and models on the backend. However, these frameworks can be complex to use, require more careful design and provision of a rich API, and do create their own issues; for instance the use of JavaScript can be a drain on the viewing devices' system resources. For this basic proof of concept only a basic website is required with limited user experience and design, and thus decoupling the frontend and the backend is not considered worthwhile.

Instead, the backend will fill pre-designed HTML templates and serve them to the browser in a more traditional way. This does increase server load but, as mentioned, increases coupling and reduces overall system complexity.

As originally specified, the backend will be based on a Python application. Python is a high level interpreted programming language which is well supported in web development (with various web frameworks) and with many well maintained libraries (for instance, the ECDSA library). The version used will be the current Python 3 (2.7 was recently deprecated) which has binaries for most modern operating systems (Linux, Windows MacOS). Python web services can be developed locally (eg. on the developer's machine) and can also be deployed to production servers. For this project it is not intended to deploy the application to a server; although the web frameworks discussed later do support WSGI[3] and so with a full CI/CD pipeline, automated deployments are possible.

In order to efficiently develop a web application with both a basic API and serving webpages, a web framework will be used. A short analysis of two options is provided:

Flask

Flask is a lightweight web application framework which supports the WSGI and can thus be easily deployed with most common web servers (Nginx, Apache) [63]. Whilst Flask does provide support for Jinga (a Python templating engine) and thus can easily support the rendering of HTML views on the backend, being a more lightweight and "lower level" framework, Flask provides a significant latitude to the developer allowing quick and easy development, however, potentially at the expense of allowing the cultivation of poor software engineering practices. It is therefore important to understand the full set of features provided by Flask, it's design patterns and the Pythonic approach - this is all covered in the documentation [63].

[2]Lightweight Directory Access Protocol: implementation of a federated login system

[3]Web Server Gateway Interface - a standard for how web servers work with the Operating System to communicate with the actual web application

Django

Django is a higher level more feature rich web framework. By default it provides tools such as a database abstraction layer, form validation, authentication controls etc. These provide a lower barrier to entry and are well maintained within the ecosystem, reducing the boilerplate required to get started and enforcing best practice and security by default. However, the prescriptive nature of the framework requires you to design your webservice in a specific way and use specific Django features. As mentioned, this can pay off in the long run by providing better maintainability, support and more predictable code (through use of a standard set of libraries) but does restrict development to the "Django way".

For this proof of concept I will use Flask. This will provide a very simple base on which to integrate my classes and although it will require writing additional boilerplate (eg. for a login system), as this is a proof of concept scope creep can be avoided through a clear design from the outset.

Webservice UML

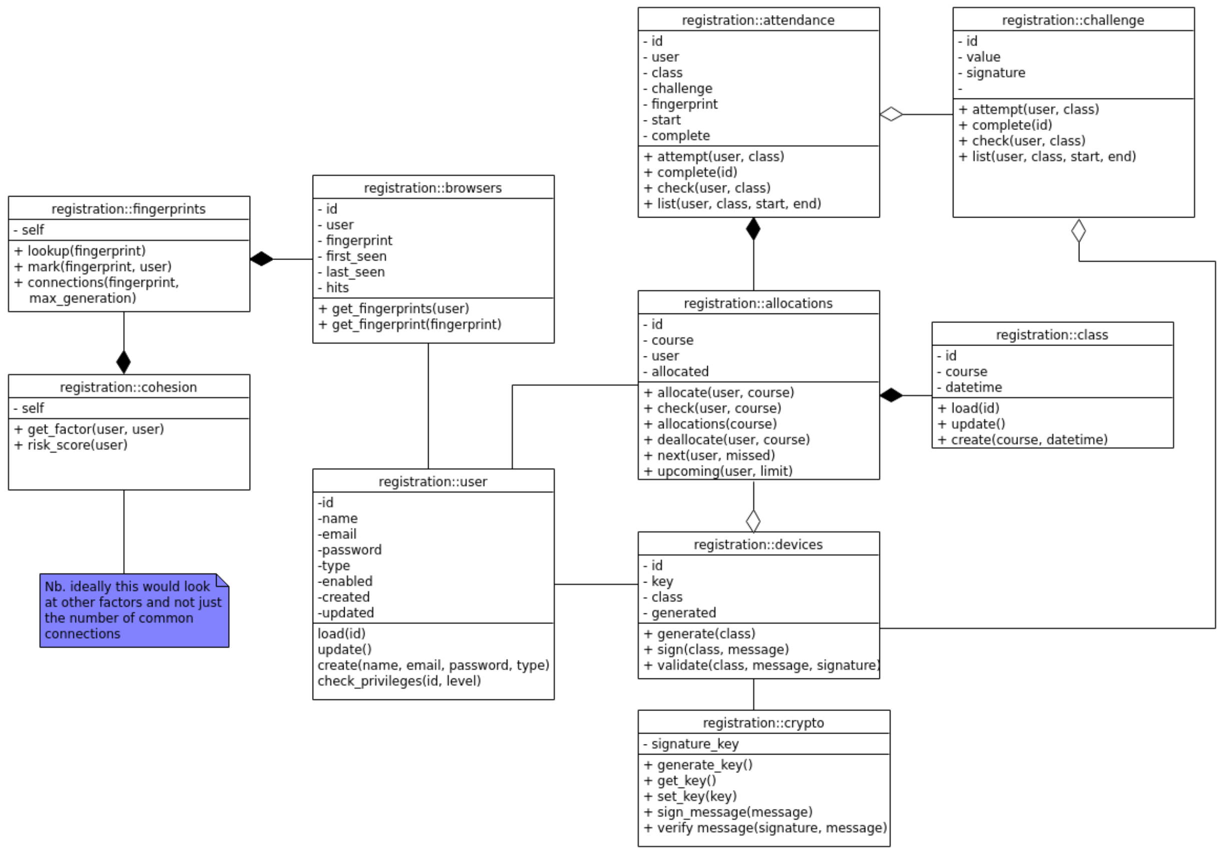


Figure 18: UML diagram. This does not include classes generated or required by Flask - only application classes.

Webservice Pages

The following pages will be provided at the following URLs as an HTML user interface, as discussed above.

/login

Form with entry for email and password to login.

/signup

Exists purely for the PoC, to allow an easy method to add new users. Will allow setting of all parameters including privilege level

/dashboard

Will display different data depending on privilege level. - Students will see their current overall attendance percentage, upcoming lectures and missed lectures - Lecturers will see their classes and attendance percentages for each class, their upcoming lectures and attendance percentage for recently held lectures - Administrators will see students with high levels of poor attendance

/register/{class_id}

Only accessible to students, starts the challenge-response flow and redirected to from their dashboard.

The frontend JavaScript is discussed in more detail later on, however as an overview an asynchronous request will be made to the backend to start the process and obtain the backend cryptogram. The cryptogram will then be signed and returned by the Bluetooth device and the result submitted by POST redirection to an endpoint for validation.

/provision/{class_id}

Only accessible to lecturers, this will generate the certificate bundle needed by the Bluetooth device for operation in that lecture.

/create/class

Only accessible to administrators, this allows the creation of classes for a specific time, against a course code.

/assign/student/{user_id}

Only accessible to administrators, this allows a student to be assigned to a course code.

/assign/lecturer/{user_id}

Only accessible to administrators, this allows a lecturer to be assigned to a course.

API

A basic API will be provided to facilitate the frontend asynchronous requests. It may be useful to provide some other endpoints, but these are not in scope for now.

/api/authenticate

- Method: POST
- Parameters: Client ID, Secret
- Returns: Access Token, Expiry

/api/registration/start

- Method: POST
- Parameters: Client ID, Access Token, User, Course, Browser Fingerprint
- Returns: Challenge ID, Challenge

For form submissions, session cookies will be checked upon submission to the endpoint at prefix **/api/forms/**.

Design Patterns

The modular unit tested design of the webservice codebase lends itself well to the use of standard design patterns. The choice of pattern for a number of components is discussed below, identifying the tradeoffs between different types of design.

One important distinction to make here is that although the webserver is constantly "spinning", that is waiting for a new connection, each request would be handled in a separate thread (with a WSGI configuration). Thus each connection is stateless and some design patterns, such as the observer pattern, do not fit well into the specific application here. An example of the observer in this context might be when interfacing with an external system and waiting for a response or webhook to service the original request, but this use case is not relevant to this codebase.

ORM/Database Class

The database class will provide a wrapper over the underlying ORM modules to add custom functionality as well as a more consistent user interface. Whilst on the surface this might look like an adaptor design patter, it will more closely follow that of a singleton. There should only ever be one database connection per session (otherwise you end up with crippling scaling issues as I discovered in the Second Year Team Project) and therefore there should only ever be a single instance of the database connection which will be shared between classes globally. The connection is effectively stateless (concurrent requests are threaded in the background) and will only ever be instantiated once, at the beginning of the session, and destructed at the very end.

It is important to note that the underlying ORM (SQLAlchemy - as discussed below) is a bridge - allowing multiple underlying database technologies to be used (eg. MySQL) with a choice of database connectors (eg. PyMySQL).

`fingerprints/cohesion`

The `cohesion` class is composed of the `fingerprints` class, which is ultimately composed of the `browsers` class. Both `cohesion` and `fingerprints` instantiate a large number of classes in the background to provide functionality - the `fingerprints` can generate a graph of fingerprint connections between users which instantiates a number of `browsers` instances. The `cohesion` class carries out some (basic) risk scoring on a user and therefore relies on the generating multiple `fingerprints` classes to calculate linked risk scores between users.

The obvious choice here might be a factory, since both classes orchestrate the creation of multiple classes behind the scenes. However, since the creation of objects is essentially in a tree structure and designed so that the eventual output is a manipulation of the entire tree structure, the composition design pattern is a better fit. Primarily this pattern helps to handle distinction between leafs and nodes in the tree, which will be an important consideration here.

`crypto`

The crypto class provides a wrapper to a more complex and extensive class. The wrapper provides a consistent set of interfaces that encapsulate several calls to the underlying class, as well as reducing the complexity of the interface. This is a classic use of the adapter pattern.

Frontend (JavaScript)

The frontend will be based on the proof of concept programs developed previously. Two major pieces of functionality are planned:

- Browser Fingerprinting
- Bluetooth device functionality

The browser fingerprint will be calculated using the ClientJS library as it provides a more stable result (based on limited testing).

The Bluetooth device functionality will be developed using the experiments in the Web Bluetooth API and the device ID will (for now) be hardcoded in the frontend to the specific development device in use.

Device

The Bluetooth device will be based on the Python ECDSA wrapper proof of concept, as the backend alongside the basic serial connection to the Bluetooth chip. It will read and verify the certificate bundle, import it and sign any data sent to the Bluetooth chip before returning it.

For MVP purposes, the Bluetooth device will sign and return any data sent to it. This may be improved to perform some kind of validation, time permitting.

Database

The application data will be stored in a relational database - this type of database lends itself well to the structured nature of the application with indexed columns.

The database engine I shall use is MySQL given my familiarity with the technology, the relative ubiquity and good library support. MySQL does not scale or lend itself to replication well but this is not required for the MVP. The schema could be easily migrated with readily available tools to another engine, such as Postgres. MySQL version 8 is the most recent stable release and whilst there are no plans to use any of the

I have elected to use an ORM (Object Relationship Mapping) tool - SQLAlchemy - with PyMySQL as the database connector.

Entity Relationship Diagram

This is not provided right now, since it essentially maps bijectivly to the UML class diagram by design.

Building the webservice

The first step in building the webservice was to configure the Python environment and set up the database. For final delivery of the code, I have decided to Dockerise the entire webservice to aid running of the code in the future and ensure a watermarked version of packages and dependencies are stored, so breaking changes introduced later do not render the software unable to run.

I first created a very naive Flask webservice which could load a page from an individual "service". I then added a database integration with a single (again naive) unit test as a proof of concept - this allowed testing of the DBMS and local connection. Creating the first version of the schema required modelling various constraints on the database to avoid consistency and normalization issues later on. For example, some tables exist purely to map IDs to IDs to remain in 3NF - this approach is discussed later on.

Having a working database, I looked into using my ORM connection to run queries on the database. Initially, I had planned to extend my existing class but I discovered there is actually a Flask plugin for SQLAlchemy and its use is encouraged as part of usage with Flask Blueprints [64].

Flask Blueprints

The naive way to build a Flask application is to have an `app.py` which serves as a kind of edge proxy - you import all of your services into this single file and bind instantiations to a URL handler. This becomes unmaintainable very quickly, can create circular import issues and makes it hard to reason about how you should deal with object classes lower down the call stack.

Flask Blueprints allow us to separate out groups of pages that go together into Pythonic modules - these share templates and associated business logic. Underlying classes and objects remain shared throughout the application as appropriate. The routing for a Blueprint works in a similar way to the naive approach, except we handle more of the subsequent request handling in the module with the associated business logic. It also allows us to use the factory design pattern with the `create_app` functionality built into Flask, which will then orchestrate the creation of the required modules to serve the request - for free! This gets rid of a lot of boilerplate that would otherwise exist to route requests.

We also are able to simplify unit and integration testing with this approach by passing different parameters to the `create_app` call to set up different objects. This avoids the situation where we unit test underlying classes, but not business logic in the modules where it is intertwined with display logic. In this case, we'll continue using pytest but use the Flask plugin to simplify the testing by providing Flask specific functionality.

In this application we will split the application out into several modules:

- `Login` to handle the login and session creation
- `Student` for all student pages and business logic
- `Lecturer` for all lecturer pages and logic
- `Administrator` for the back office processing pages and logic
- `API` for the basic API we'll be providing

The implementation of the Blueprints was straightforwarded and greatly simplified the structure of the application by handling creation of objects within the `create_app` call, which are then passed down to Blueprints in the app context. I started by creating two blueprints initially - for handling login, signup and the student dashboard - with templates and the forms for those pages.

I have elected to use `flask_bootstrap` which is a Flask library which handles the importing of the well known Boostrap 3.7.X library for UI components. This isn't the latest version of Bootstrap, but it does integrate well with other Flask libraries like What The Form.

What The Form (`flask_wtf`) is a form creation and validation library. Forms are defined centrally (so they can be reused) and instantiated then served within the Blueprint application logic. Frontend and backend validation are handled by the library as well as CSRF protection, with the results then passed to the application for processing.

ORM and Testing

Work on integrating the ORM and test harness did not go as planned. Despite a couple of weeks' work and debugging alongside various examples and documentation, it was not possible to get either to work as expected.

SQLAlchemy expects the models for tables and objects to be defined in Python. Within the application logic you specify for each object which fields are required and how they related to one another. This maps fairly closely to tables and relationship contrsaints at the SQL layer. However, you do not get an SQL output. Each time the application is run you should provide a configuration file (usually passed as an argument or switch) which defines the connection to a raw database. The ORM will then either drop all the tables and create the schema as defined in your Python models (for a staging environment) or validate the tables that exist in the database are as it expects (for production where you want data to persist between restarts). Unfortunately, by using MySQL and writing the schema first in SQL this meant using SQLAlchemy would have required rewriting the entire schema in Python - a non-trivial effort. Given my unfamiliarity with the technology and later experiences with the test harness, I later opted to fallback to directly writing SQL through the connector (PyMySQL) within object classes.

The test harness is a Flask specific version of Pytest (`flask_pytest`). This is designed to hookin more closely to Flask and allow running unit tests against the defined Flask routes. In Flask applications you tend to have object classes which represent standard "real world objects", as is common in OOP, as well as classes to provide specific sets of functionality (eg. a class to handle processing payments). Within routes for each page there is then application logic which "orchestrates" the calling of various object methods; for example, on the login page, on form submission the route application logic loads the user by email and then calls a method to verify the password provided. This application logic in the routes is untested unless you have integration and front end unit tests which call the Flask routes directly. Unfortunately, getting this to work properly is non-trivial. Similarly to ORM, a configuration file is passed to set up the test environment which itself expects to instantiate and pass the database object to the running application. This I still do not quite understand, as it makes more sense for logic to handle instantiating databases to remain within the application and for the tests to simply configure the application into a staging environment. It is also unfortunate that by design these tests are very close to the user interface and therefore need to process the raw HTML - therefore caring more about where data is, than what the data is. Trying to get these flakey tests in a complicated framework to run took a significant amount of time and was not making progress. Therefore, in order to prioritise producing working code over comprehensive tests and documentation, I decided to adopt a different approach.

As previously mentioned, I switched from using an ORM to writing raw SQL in the classes. This does make the code less robust and it does mean that changes to either the schema or class functionality may require more cascading changes. This is mitigated by proper OOP discipine and good schema design, so a single object is represented in a class, mapped to a single a table (eg. a user). Where you need to extend or adapt classes, proper design patterns (such as an adaptor pattern to provide a custom interface) that build on core objects are used, that also keep the database interactions for a single table within a single class. By adopting this design methadology, it does require more boilerplate code to provide functionality that you would otherwise get "for free" with an ORM, for example joining classes using known relations. In a small project this benefit is fairly negligible. There are no security risks with this change: both an ORM layer and directly interfacing with the connector used prepared statements which minimise the risk of SQL injection.

The testing approach was now to use Pytest to directly test individual classes, but not carry out extensive integration testing or any automated UI testing. This carries a risk in that the route application code (which orchestrates calls to core classes) will not have automated testing but is partially mitigated by comprehesive testing of underlying classes and smoke testing during development. It is accepted that bugs will get past this so the idea is to minimise their impact by ensuring underlying classes are robust - it is better to raise an unhandled exception because of bad application logic than for data intrgrity to be comprimised, or inconsistentencies in class structures to develop. By following Test Driven Development principles I ensured proper test coverage and an appropriate granularity of tests were provided.

Building the signup and login logic

These components depend on the `user` class. This provides an abstraction of the user object with various helped methods to load a user, check the password, create a new user etc. The class is stateful, as expected in OOP.

The login functionality was built first and utlised the `flask_login` library. This library abstracts handling cookies and sessions away, just requiring some hooks into the user class. A decorated function is provided within the application context which allows `flask_login` to load a user ID and check several properties in the object. These properties are set dyanically depending on the internal status of the object. For exmaple, the `is_active` property is used to test whether the user account has been enabled or not. When the user login has been sucessfully validated, the `login_user` function is called which causes `flask_login` to issue session cookies before the application redirects the user to a protected page.

One property required is named `is_authenticated`. Intuitively and as implied by the `flask_login` documentation [65] this is set to True when the user authenticates with the application, and False any other time. In practice, this actually controls whether a user is considered "anonymous" or not, even if the `is_anonymous` property always returns False. I am working with the maintainers to make this clearer, or change the behaviour to make it clearer how `flask_login` expects the properties to be used [66].

With login working, I needed to implement permisions for loading pages - you do not want a student accessing a lecturer's administration page! There are various Flask plugins for this - `flask_allows` and `flask_principal` are two. Unfortunately, I experienced significant issues with interfacing these with `flask_login` - it transpires that the `flask_security` project actually provides a full suite of tools that include both `flask_login` and `flask_principals` working together; I would in future explore this library before developing my own independent integrations. I have instead elected to provide a simple property in the user class for application logic to check the type of user that is trying to access the page. Since tiered access control is not required, this simple solution actually avoids the need for complex external libraries and provides a cleaner codebase.

Creating users is a relatively simple affair with the form being marshalled into the relevant methods of the user class to create the account. Errors such as a duplicate email are handled and provided to the frontend, but not yet displayed - as with login errors. This requires a default template and inclusion of the relevant HTML to display errors on any page. This is another feature provided by Flask out of the box and stores the error message in the session cookie to prevent XSS attacks.

Adding additional functionality

Work continued using WTF and additional classes to add functionality to add students to a course and to add lectures to a course. This was supplemented with visual changes for an administrator and a student to display these new relations.

Unit vs Integration Tests

Originally I intended to write full unit tests for the project. As mentioned in the "ORM and Testing" section above, originally the aim was to use the inbuilt Flask testing mechanisms but it was not possible to get this fully working. Instead, as mentioned, I decided to use `pytest` itself and write tests for the individual classess. Because these interacted with the database they tended to be more abstract, testing more functionality and doing so against a live database. I have subsequently realised I have written integration tests, as opposed to unit tests. In order to unit test the classes I would need to mock the database connection which was beyond the scope by this point.

Delivery

The delivered software was not envisioned. Several major items produced in the proof of concept were never implemented, as they weren't needed and could not be used without the main component: the Web Bluetooth API. More detailed conclusions are drawn from this in the reflective areas of the report.

Implemented:

- Boilerplate and Flask Blueprints
- Login / Signup
- Student/Administrator pages
- Creation of courses and assignment of students to lectures

Not implemented:

- Registration
- Fingerprint tracking

Professional Issues

Privacy and freedom of expression is becoming an increasingly debated issue, especially online and in the digital world. As computing power and storage capacity have increased over the last few decades, it has become feasible for companies to collect large amounts of data at an individual level for analysis and data mining. Whilst often the data is claimed to be anonymised, studies such as [67] have shown that it is possible to use modern machine learning techniques on large datasets to identify individuals.

This project advocates not only requiring students to prove their presence in an auditable way but the bulk collection of browser data to prevent fraud and deception. Taking the former as a given, is it proportionate to collect a uniquely identifiable hash of the student's browser each time they mark their attendance?

One important consideration is data protection, specifically the requirements of the General Data Protection Regulation or GDPR. As this data can uniquely identify a browser, it is possible it could be used in conjunction with other information (for example, lectures attended) to identify an individual. It is therefore classed as Personally Identifiable Information.

In order to process the data we must have a lawful reason to do so - in this case we claim legitimate interest applies. We must therefore meet the following test:

> Identify a legitimate interest; Show that the processing is necessary to achieve it; and Balance it against the individual's interests, rights and freedoms. [68]

Our legitimate interest is in the prevention of fraud but we still need to show that our processing of the PII is necessary, proportionate and balances the individual freedoms of the person.

This continues:

> Does this processing actually help to further that interest? Is it a reasonable way to go about it? Is there another less intrusive way to achieve the same result?

The processing of the PII does further the interest as we use it to measure connections between students and to identify suspicious patterns of behavior (eg. a single device signing in multiple students). We consider this to be a reasonable method of measuring this data points and that this is the least intrusive method - we do not store the raw data for example, just the irreversible fingerprint hash.

The ICO test around individual freedoms look mainly at the sensitivity of the data, disclosure of the processing and appropriate safeguards to minimize harm. The data is fairly sensitive - appropriate controls should be in place to limit and record access and it should be removed as soon as the legitimate interest ends.

Even though we can demonstrate that the data collection in this scenario is proportionate, there are a number of other uses for browser fingerprints, particular in advertising, which operate in a decidedly more grey area. This type of tracking is not uncontroversial and from 2012 to 2014, Verizon (a US network carrier) injected unique identifiers into network traffic without their customers being aware of such 'supercookies' being attached to their data. [69] [70] It transpired that not only were Verizon using these supercookies themselves, but third parties had discovered their existence and were using them to track individual devices for purposes such as advertising. The FCC's investigation determined that Verizon should have sought explicit opt-in consent from customers for the direct sharing of what the FCC referred to as UIDH (Unique Identifier Headers) and given the option for customers to opt out of their use by Verizon internally. A specific case cited by the FCC related to a third party advertiser using supercookies to continue tracking customers after they had explicitly removed normal cookies from their devices.

As previously discussed, the GDPR applies within the UK and contains specific provision for what it calls "Special Category Data". This is "personal data which the GDPR says is more sensitive, and so needs more protection." [71] and includes "biometric" data which traditionally has been used to refer to specific human characteristics (such as retina data). This could arguably be applied to specific characteristics of a device a user owns; in the same way that an IP address is considered Personally Identifiable Information.[72]

In an increasingly digital world, where the resources exist to store large volumes of data for an indeterminate period and carry out increasingly accurate machine learning and modelling, the practice of trying to uniquely identify a user across the internet by their browser fingerprint is concerning. Not only does this allow private corporations another alarming way to build up data profiles of citizens, without their knowledge, but nation-state actors can also use this highly targeted and specific data to track down individuals and groups. Law enforcement have long argued this kind of unique identifier allows them to catch criminals - but at what level of accuracy and at what cost to individual liberties and freedoms? It is easy to dismiss these concerns as "fanatical" or "out of touch with reality" whilst we live in a society where freedom of speech is championed and the rights of an individual protected. As the Metropolitan Police Service in London begin their rollout of facial recognition technology, it is clear this issue will continue to be debated in various forms for a while to come. [73] [74]

To ensure compliance with the GDPR in the proposed system and to protect student data and privacy, several considerations have therefore been taken into account:

- Additional measures will be taken to ensure that browser fingerprints are not accessible to users other than administrators. In a production system additional audit logging would take place here, but this is considered out of scope.
- Administrators will be given the option to purge PII for a given user from the database or to remove all records from the database - such as at the end of the academic year. This will, unfortunately, remove non-PII (such as the attendance data) associated with a user due to database integrity constraints. A workaround is for an adminstrator to bulk export data beforehand, but a technical solution to this is out of scope.

Assessment

Outcomes

In the project, I looked at a wide range of topics within Computer Science, focussing on the hardware and design behind attendence monitoring solutions. Firstly, I successfully implemented (using code from Nick Mooney [8]) a clicker basestation and associated Python script which I believe actually has a practical use, to replace the current PowerPoint slide system. The emulation of the clickers themselves proved more tricky and the adaptation of the code to spoof packets has been difficult. However, the reception and decoding of live packets has shown that the protocol is vulnerable to a theoretical programmatic attack and, if nothing else, is vulnerable to a simple replay attack of a recorded transmission. I then went on to explore the practicalities of browser fingerprinting, the libraries that are available and what attributes of a browser allow it to be uniquly identifiable. This aspect of my project was further explored in the Professional Issues section, looking at the legitimate interest test under GDPR for PII and changes that could be made to the project to further enhance the security of personal data.

A key part of my project looked a potential solutions for the clickers. One idea explored was the student ID cards, based on MIFARE Classic technology. The known flaws in the technology were discussed and a practical attack against the student ID cards demonstrated. Focus then turned to Bluetooth technology and specifically working with Bluetooth Low Energy devices and getting to work with the Web Bluetooth APIs. This has then extended to the development of a basic and scoped MVP to demonstrate the practical benefits of a solution designed on the basis of research in this project.

The goal of the software MVP was to show a web based authentication system that allows students to securely register their attendance at a lecture. This ultimately did not materialise due to a combination of factors: time constraints, the impact of a worldwide pandemic and issues in getting the authentication step with Web Bluetooth (latterly Web USB) working.

I did, however, build a very basic MVP in Flask using a number of new technologies to me, including Flask Blueprints, Flask Login, Flask Access and Docker, to name a few. Although this never was completed due to the issues in making the crucial part of the system work, it does lay the foundation for a more complete system if those issues can be overcome in time.

Issues encountered

Some of the earliest issues with the project have focused around the hardware. Working with hardware is always more challenging than with software - it introduces a level of complexity not limited to operating system drivers and hard to debug code running on microcontrollers. Some of the earliest issues I experienced were with the nRF24E1 devices and the correct pinout for the device - it is not just “plug and play”. I later struggled with bugs in encoding in the interface between two languages and three devices. Ultimately, I did not succeed in getting this to work but I did learn a considerable amount about hexadecimal encoding in C++ and historical encoding problems in terminals and shells.

Again, hardware issues were encountered with the HM-10 devices which appeared not to follow the specification properly. Documentation was very limited and hampered troubleshooting for a number of weeks. It later transpired that the devices themselves did not follow the specification properly and by purchasing a new device from an alternative source, I very quickly got results. Orthagonally, I think this goes against much of the ingrained self-doubt around the quality of ones code and it is important to switch out all variables when debugging, instead of making the assumption that a commercially available black box is functioning as expected.

The Web Bluetooth API is another demonstration of the same. Although the documentation is well written and sets out a number of exciting features, it suffers from a lack of W3C support and only having three maintainers. As such, it only works in development mode in Google Chrome and not with any drivers I have tried. Although some of the contraints were known beforehand, the full extent of the limitation was not clear until actually tested. I believe with another few months working full time on the project, it would be possible to patch the library to a state where it is more usuable - this is an interesting future extension of the project as the wide applicability of the Web Bluetooth API is a compelling reason for future work on it.

Other issues stemmed from bad documentation. `flask_login` for example misrepresented a property which if not permanently set to `True` results in no user being able to log in. This seems to actually be a bug in the implementation, but as it is a breaking change, I have proposed a modification to the documentation [75]. I also experienced issues in using new technology with Flask. Whilst I wanted to make my Flask projects more scalable and use proper design patterns with Flask Blueprints, I did have issues introducing a hollistic set of unit tests including the ORM and database connection. The lesson learned here is to avoid using too much new and unfamilier technology - Flask Blueprints were already a steep learning curve and by trying to use a new style of unit testing and a new ORM with dynamic configuration in addition, the risk of nothing working at the end dramatically increases.

Another problem related to the `pytest` integration tests. These were originally intended to be unit tests but they gradually morphed into integration tests run against the live database. As mentioned, in order to write full unit tests the connection to the database should have been mocked. Learning how to do proper Python unit testing with the Flask webserver and a database connection is something I would look into further before starting the project, as it would have made the code easier to verify and more robust. Having automated testing has been useful and this serves to reinforce the crucial nature of complete test environments.

When comparing these issues to the original risk assessment (Appendix 7), the risks identified did all come into play, to various degrees. Of course the risk that ocurred with the largest impact relates to the Bluetooth device comunication, which ultimately has not worked reliably. Alternatives, such as Web NFC were investigated and discarded, whereas Web USB was investigated and trialed but further technical issues and time constraints in investigating these meant it was not ultimately implemented.

Other risks, such as with the clicker analysis and functional design have manifested, but not to the same degree as the impact of the issues with the web APIs. Having all the assessed risks in mind has helped in planning the rest of the project, helping to avoid disproportionate amounts of time being given over to small areas of the project. This has meant no large blockages in work built up, but has had a material impact on deliverables to date.

Conclusion

The project has achieved it's broad goals. Work has been done to show the clickers are vulnrable to simple attacks, and a system using modern technologies has been designed and partially implemented. Ethical issues surrounding browser fingerprinting and privacy in general have been explored, as has the depths of the Web Bluetooth specification. Research into the various proof of concenpt systems has been successful and lays the groundwork for further developments in this area. A significant amount of time has been spent develping the software deliverable - a Flask application - and whilst it did not meet the envisioned goals, it did go a way to demonstrate the holistic concept of the system.

Self-evaluation

Overall this has been an interesting and engaging project which has given me the opportunity to explore a wide range of topics within Computer Science, and even those with a wider focus such as the application of the GDPR and some of the electronic engineering elements.

The research areas have been very rewarding and provided siginificant context in carrying out the project. Whilst there has not been a large amount of literature on really any of the topics researched, the material available has been of good quality and useful. I have enjoyed reading complex specifications and digging into documentation, and the research elements of my project have been amongst the most enjoyable aspects of the project.

It is very easy to generalise failures in time managment, so instead I will endeavour to provide a more comprehensive explanation of some of the less successful areas of the project.

The first problems came with hardware issues. Overall, I think the use of so many hardware components was too ambitious in the time available and led to parts of the project being compressed and others expanded. For instance, the HM-10 issues took several weeks to resolve which had a knock on impact with the rest of the project. Had I consulted Dave Cohen earlier I might have resolved this particular issue more quickly, but it was far from the only problem!

The Web Bluetooth API turned out to be a very immature interface to work with, and I was ultimately unable to suceed in getting it working with several devices. To understand and fix these issues would involve learning how to develop parts of the Chromium Project which is not achievable in the time period. I think had the scope been limited to building a simple application to work with the Web Bluetooth specification, it would have been possible to get this working. I later looked at the Web USB API as a replacement - a similar but crucially W3G specification. I got further with this endeavour, but again did not succeed. One key outcome is that whilst the documentation may look good, there is probably a reason why there are not many (if any) third party projects using a technology!

The delays with hardware problems led to knock on delays in working on the larger software deliverable. This is regrettable since it formed an interesting and large part of the overall project. However, it would never have been fully functional without the technologies explored in the earlier phases. Again, had the scope been more limited then this would have been more achievable. I did however learn more about using Docker and Docker Compose when preparing the project for submission, which was useful.

Another interesting learning vector has been the combination of Pandoc and LaTeXthat has generated this document. Earlier reports and version of this report were in two column format; the reason for the switch is again in immature software - Pandoc does not easily support tables spanning columns although a partial implementation has been proposed as of late March! [76]

It is impossible to ingore the elephants in the room: strikes and COVID-19. These have had a material impact on my project, affecting meetings with my supervisor before Christmas and in the Spring, and the ultimate delivery of the project after the United Kingdom entered lockdown in the final weeks of term. It is impossible to predict a global pandemic which has made elements of my project harder, notably the dissemination of hardware to markers.

Software Deliverables

Several deliverables are provided. Basic installation instructions can be found in the User Manual in Appendix 11. Outputs from tests are found in Appendix 9.

Docker

I originally experiemented in using Docker to package the Arduino dependencies and build environment. However issues with missing system libraries for timing within the container rendered this not possible.

The Flask application was Dockerised using a number of guides. Initially a Dockerfile for just the Flask application was made using online guides [77] [78] with only a few issues relating to binding the Flask development webserver to the correct host. In order to orcestrate the start of another container, the MySQL database, it was necassary to use `docker-compose`. Initially, following another blog post [79] there were issues in getting the application to connect to the database. I understand from the aforementioned official documentation, it is important to refer to hostnames (eg. `web` and `mysql`) within containers so that the Docker DNS can dynamically provide IP addresses. There were also various credential issues and a strange InnoDB problem with the schema: when you start the containers you will see a log line `Tablespace \registration\.\browsers\' exists..` which causes that table, but no other, to not be created. This was not resolvable in itself, but by rearanging the order of the schema file, an unused table was sacrificed instead of one that was required.

Acknowledgements

Thanks to Tom Pollard [80] for the front cover template which I have adapted, “corentin” for the Gantt chart example in LaTeX[81] and Dave Cohen [82] for the Final Year Project guide and suggested layouts.

I owe a debt of gratitude to my supervisor, Hugh Shanahan, who has been invaluable in providing advice even in the most difficult of situations, when it seemed nothing would work!

References and Bibliography

[1] Home Office (UK Government), “Tier4 of the Points Based System: Guidance for Sponsors Document 2: Sponsorship Duties.” Aug. 2019, [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/824003/Tier_4_Sponsor_Guidance_-_Doc_2_-_Sponsorship_Duties_2019-08_1.1.pdf.

[2] Royal Holloway Department of Computer Science, “DEPARTMENT OF COMPUTER SCIENCE UNDERGRADUATESTUDENT HANDBOOK.” Sep. 2018, [Online]. Available: https://intranet.royalholloway.ac.uk/computerscience/documents/pdf/currentug/ug-student-handbook-2018-19.pdf.

[3] Universities UK, “THE STUDENT VISA SYSTEM: PRINCIPLES TO REFORM.” Jul. 2019, [Online]. Available: https://www.universitiesuk.ac.uk/policy-and-analysis/reports/Documents/2019/student-visa-system-principles-to-reform.pdf.

[4] EY, “Challenges and costs of the UK immigration system for Russell Group universities.” Mar. 2019, [Online]. Available: https://russellgroup.ac.uk/media/5750/challenges-and-costs-of-the-uk-immigration-system-for-russell-group-universities.pdf.

[5] T. Goodspeed, “Travis Goodspeed’s Blog: Reversing an RF Clicker,” *Travis Goodspeed’s Blog*. Jul. 2010, Accessed: Sep. 17, 2019. [Online]. Available: https://travisgoodspeed.blogspot.com/2010/07/reversing-rf-clicker.html.

[6] Nordic Semiconductor ASA, “Single chip 2.4 GHz Transceiver - nRF2401.” Jun. 4AD, [Online]. Available: https://www.sparkfun.com/datasheets/RF/nRF2401rev1_1.pdf.

[7] C. Borrelli, “IEEE 802.3 Cyclic Redundancy Check.” Xilinx, Mar. 2001, [Online]. Available: https://www-inst.cs.berkeley.edu/~cs150/fa04/Lecture/xapp209.pdf.

[8] N. Mooney, “Nickmooney/turning-clicker.” Nov. 2016, Accessed: Sep. 17, 2019. [Online]. Available: https://github.com/nickmooney/turning-clicker.

[9] Frank, “FrankBoesing/FastCRC.” Jun. 2019, Accessed: Oct. 25, 2019. [Online]. Available: https://github.com/FrankBoesing/FastCRC.

[10] “1/2/3/5/10 Arduino NRF24L01+ 2.4GHz Wireless RF Transceiver Module UK Seller,” *eBay*. Accessed: Oct. 25, 2019. [Online]. Available: https://i.ebayimg.com/images/g/rkQAAOSw7k9dHxoc/s-l400.jpg.

[11] “nRF24L01+ Transceiver Hookup Guide - learn.sparkfun.com.” Accessed: Oct. 15, 2019. [Online]. Available: https://learn.sparkfun.com/tutorials/nrf24l01-transceiver-hookup-guide.

[12] “LCD Clicker Response Devices,” *Turning Technologies*. Accessed: Sep. 17, 2019. [Online]. Available: https://www.turningtechnologies.com/lcd/.

[13] “Arduino Reference.” Accessed: Oct. 28, 2019. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/communication/serial/readstringuntil/.

[14] D. M. Kristol and L. Montulli, “HTTP State Management Mechanism.” Accessed: Nov. 14, 2019. [Online]. Available: https://tools.ietf.org/html/rfc2109.

[15] “Panopticlick About.” Accessed: Nov. 14, 2019. [Online]. Available: https://panopticlick.eff.org/about.

[16] K. Boda, Á. M. Földes, G. G. Gulyás, and S. Imre, “User Tracking on the Web via Cross-Browser Fingerprinting,” in *Information Security Technology for Applications*, 2012, pp. 31–46, doi: 10.1007/978-3-642-29615-4_4.

[17] Mozilla, “Browser Extensions,” *MDN Web Docs*. Accessed: Nov. 15, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions.

[18] A. Kobusińska, J. Brzeziński, and K. Pawulczuk, “Device Fingerprinting: Analysis of Chosen Fingerprinting Methods:” in *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security*, 2017, pp. 167–177, doi: 10.5220/0006375701670177.

[19] Hovav Shacham and Keaton Mowery, “Pixel Perfect: Fingerprinting Canvas in {HTML5},” *Proceedings of W2SP 2012*, May 2012, Accessed: Nov. 15, 2019. [Online]. Available: https://hovav.net/ucsd/papers/ms12.html.

[20] M. Ashouri, “A Large-Scale Analysis of Browser Fingerprinting via Chrome Instrumentation,” p. 12, Jul. 2019.

[21] “Device fingerprinting Adyen Docs.” Accessed: Nov. 13, 2019. [Online]. Available: https://docs.adyen.com/risk-management/device-fingerprinting.

[22] V. V, “Valve/fingerprintjs2.” Jan. 2020, Accessed: Nov. 15, 2019. [Online]. Available: https://github.com/Valve/fingerprintjs2.

[23] “ClientJS.” Accessed: Nov. 13, 2019. [Online]. Available: https://clientjs.org/.

[24] “C21 Clicker.” Accessed: Oct. 30, 2019. [Online]. Available: https://www.c21technoventures.com/clicker.php.

[25] P. Nikitin and K. Rao, “Performance limitations of passive UHF RFID systems,” in *2006 IEEE Antennas and Propagation Society International Symposium*, Jul. 2006, pp. 1011–1014, doi: 10.1109/APS.2006.1710704.

[26] J. Fischer, “NFC in cell phones: The new paradigm for an interactive world [Near-Field Communications],” *IEEE Communications Magazine*, vol. 47, no. 6, pp. 22–28, Jun. 2009, doi: 10.1109/MCOM.2009.5116794.

[27] R. Heydon, *Bluetooth low energy the developer’s handbook.* Upper Saddle River, N.J.: Prentice Hall, 2013.

[28] K. E. Mayes and C. Cid, “The MIFARE Classic story,” *Information Security Technical Report*, vol. 15, no. 1, pp. 8–12, Feb. 2010, doi: 10.1016/j.istr.2010.10.009.

[29] A. Kerckhoffs, “La Cryptographie Militaire (Seconde partie),” *February 1883*, p. 33.

[30] K. Nohl and H. Plötz, “Mifare.” Accessed: Oct. 31, 2019. [Online]. Available: /v/24c3-2378-en-mifare_security.

[31] K. Nohl, D. Evans, and H. Plotz, “Reverse-Engineering a Cryptographic RFID Tag,” p. 9.

[32] “Nfc-tools/mfcuk.” nfc-tools, Jul. 2018, Accessed: Nov. 27, 2019. [Online]. Available: https://github.com/nfc-tools/mfcuk.

[33] G. Klostermeier, “Ikarus23/MifareClassicTool.” Nov. 2019, Accessed: Nov. 27, 2019. [Online]. Available: https://github.com/ikarus23/MifareClassicTool.

[34] Bluetooth SIG, *Specification of the Bluetooth System*, 4.0 ed. 2010.

[35] C. Gomez, J. Oller, and J. Paradells, “Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology,” *Sensors*, vol. 12, no. 9, pp. 11734–11753, Aug. 2012, doi: 10.3390/s120911734.

[36] K. Townsend, C. Cufí, Akiba, and R. Davidson, *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking.* O’Reilly, 2014.

[37] Hsiang-Te Ho, Hsing-Ya Chiang, and Sheau-Chien Wang, “US6480553.Pdf,” US6480553.

[38] Bluetooth SIG, *Specification of the Bluetooth System*, 4.1 ed. 2013.

[39] National and Institute of Standards and Technology, “FIPS 197, Advanced Encryption Standard (AES),” *Processing Standards Publication*, no. 197, p. 51, Nov. 2001.

[40] J. Daemen, “The Rijndael Block Cipher,” p. 45, Sep. 1999.

[41] D. Loginov, “How to make a web app for your own Bluetooth Low Energy device?” *Medium.* Jun. 2019, Accessed: Oct. 28, 2019. [Online]. Available: https://medium.com/@loginov_rocks/how-to-make-a-web-app-for-your-own-bluetooth-low-energy-device-arduino-2af8d16fdbe8.

[42] WebBluetoothCG, “Web Bluetooth.” Nov. 2019, Accessed: Oct. 28, 2019. [Online]. Available: https://webbluetoothcg.github.io/web-bluetooth/.

[43] gbaman, “Simple guide for setting up OTG modes on the Raspberry Pi Zero, the fast way!” *Gist.* May 2016, Accessed: Nov. 28, 2019. [Online]. Available: https://gist.github.com/gbaman/975e2db164b3ca2b51ae11e45e8fd40a.

[44] “Raspberry Pi RTS / CTS Flow Control – Ethertubes.” Accessed: Nov. 05, 2019. [Online]. Available: https://ethertubes.com/raspberry-pi-rts-cts-flow-control/.

[45] “The Pi4J Project – Pin Numbering - Raspberry Pi 2 Model B.” Accessed: Nov. 05, 2019. [Online]. Available: https://pi4j.com/1.2/pins/model-2b-rev1.html.

[46] JNHuaMao Technology Company, “HM-10-datasheet.pdf.” Mar. 2014, Accessed: Nov. 04, 2019. [Online]. Available: https://xdripkit.co.uk/HM-10-datasheet.pdf.

[47] M. Milosevich, “AT Commands Reference Guide,” p. 614, Aug. 2006, [Online]. Available: https://www.sparkfun.com/datasheets/Cellular%20Modules/AT_Commands_Reference_Guide_r0.pdf.

[48] “How to communicate with serial uart bouetooth module on RPi2 - Raspberry Pi Forums.” Accessed: Nov. 04, 2019. [Online]. Available: https://www.raspberrypi.org/forums/viewtopic.php?f=91&t=158856&p=1032763#p1032763.

[49] Mark Duncombe, “Is UART Flow Control (RTS/CTS) Necessary for Proper Operation in Wireless Modules?” *Laird Connect.* Accessed: Nov. 05, 2019. [Online]. Available: https://www.lairdconnect.com/resources/blog/uart-flow-control-rtscts-necessary-proper-operation-wireless-modules.

[50] Silicon Labs, “AN0059.0: UART Flow Control.” Jan. 2017, Accessed: Nov. 28, 2019. [Online]. Available: https://www.silabs.com/documents/public/application-notes/an0059.0-uart-flow-control.pdf.

[51] Vince Deater, “Raspberry Pi Hardware Flow Control.” Accessed: Nov. 05, 2019. [Online]. Available: https://www.deater.net/weave/vmwprod/hardware/pi-rts/.

[52] Martyn Currey, “HM-10 Bluetooth 4 BLE Modules.” Jan. 2017, Accessed: Nov. 18, 2019. [Online]. Available: https://webcache.googleusercontent.com/search?q=cache:3SL-aKhV5CkJ:www.martyncurrey.com/hm-10-bluetooth-4ble-modules/+&cd=1&hl=en&ct=clnk&gl=uk&client=ubuntu.

[53] Mozilla, “Web Bluetooth API,” *MDN Web Docs.* Accessed: Oct. 28, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API.

[54] F. Beaufort, “Interact with Bluetooth devices on the Web,” *Google Developers.* Jul. 2015, Accessed: Oct. 28, 2019. [Online]. Available: https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web.

[55] “Web Bluetooth API - Chrome Platform Status.” Accessed: Nov. 28, 2019. [Online]. Available: https://www.chromestatus.com/feature/5264933985976320.

[56] acassis, “How to get Chrome Web Bluetooth working on Linux,” *Alan C. Assis.* Jun. 2016, Accessed: Nov. 28, 2019. [Online]. Available: https://acassis.wordpress.com/2016/06/28/how-to-get-chrome-web-bluetooth-working-on-linux/.

[57] François Beaufort, “Interact with NFC devices on Chrome for Android.” Feb. 2020, Accessed: Mar. 08, 2020. [Online]. Available: https://web.dev/nfc/.

[58] François Beaufort, “Access USB Devices on the Web,” *Google Developers.* Mar. 2016, Accessed: Mar. 08, 2020. [Online]. Available: https://developers.google.com/web/updates/2016/03/access-usb-devices-on-the-web.

[59] “Webusb/arduino.” webusb, Aug. 2019, Accessed: Apr. 11, 2020. [Online]. Available: https://github.com/webusb/arduino.

[60] G. Kalantar, A. Mohammadi, and S. N. Sadrieh, “Analyzing the Effect of Bluetooth Low Energy (BLE) with Randomized MAC Addresses in IoT Applications,” in *2018 IEEE International Conference on*

Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Jul. 2018, pp. 27–34, doi: 10.1109/Cybermatics_2018.2018.00039.

[61] EMVCo, "EMV Contactless Specifications for Payment Systems," p. 589, 2016.

[62] T. Chothia, I. Boureanu, and L. Chen, "Making Contactless EMV Robust Against Rogue Readers Colluding With Relay Attackers," p. 10.

[63] Pallets, "Welcome to Flask — Flask Documentation (1.1.x)." 2010, Accessed: Jan. 09, 2020. [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/.

[64] Todd Birchard, "Organizing Flask with Blueprints," *Hackers and Slackers*. Sep. 2018, Accessed: Jan. 30, 2020. [Online]. Available: https://hackersandslackers.com/flask-blueprints/.

[65] "Flask-Login — Flask-Login 0.4.1 documentation." Accessed: Feb. 17, 2020. [Online]. Available: https://flask-login.readthedocs.io/en/latest/.

[66] crablab, "Forcing 'is_authenticated' True persistently · Issue #475 · maxcountryman/flask-login," *GitHub*. Feb. 2020, Accessed: Feb. 20, 2020. [Online]. Available: https://github.com/maxcountryman/flask-login/issues/475.

[67] L. Rocher, J. M. Hendrickx, and Y.-A. de Montjoye, "Estimating the success of re-identifications in incomplete datasets using generative models," *Nature Communications*, vol. 10, no. 1, pp. 1–9, Jul. 2019, doi: 10.1038/s41467-019-10933-3.

[68] "Legitimate interests." Apr. 2019, Accessed: Jan. 27, 2020. [Online]. Available: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/lawful-basis-for-processing/legitimate-interests/.

[69] J. Brodkin, "Verizon's 'supercookies' violated net neutrality transparency rule," *Ars Technica*. Jul. 2016, Accessed: Nov. 14, 2019. [Online]. Available: https://arstechnica.com/information-technology/2016/03/verizons-supercookies-violated-net-neutrality-transparency-rule/.

[70] "Verizon, AT&T tracking their users with 'supercookies' - The Washington Post." Accessed: Nov. 14, 2019. [Online]. Available: https://www.washingtonpost.com/business/technology/verizon-atandt-tracking-their-users-with-super-cookies/2014/11/03/7bbbf382-6395-11e4-bb14-4cfea1e742d5_story.html.

[71] GOV.UK, "Guide to the General Data Protection Regulation," *GOV.UK*. Mar. 2018, Accessed: Nov. 14, 2019. [Online]. Available: https://www.gov.uk/government/publications/guide-to-the-general-data-protection-regulation.

[72] "EUR-Lex - 62014CJ0582 - EN - EUR-Lex." Oct. 2016, Accessed: Nov. 14, 2019. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A62014CJ0582.

[73] V. D. Police and crime correspondent, "UK police use of facial recognition technology a failure, says report," *The Guardian*, May 2018, Accessed: Jan. 28, 2020. [Online]. Available: https://www.theguardian.com/uk-news/2018/may/15/uk-police-use-of-facial-recognition-technology-failure.

[74] F. Kaltheuner, "Facial recognition cameras will put us all in an identity parade Frederike Kaltheuner," *The Guardian*, Jan. 2020, Accessed: Jan. 28, 2020. [Online]. Available: https://www.theguardian.com/commentisfree/2020/jan/27/facial-recognition-cameras-technology-police.

[75] crablab, "Change to better describe 'is_authenticated' by crablab · Pull Request #476 · maxcountryman/flask-login," *GitHub*. Feb. 2020, Accessed: Feb. 26, 2020. [Online]. Available: https://github.com/maxcountryman/flask-login/pull/476.

[76] "Better tables by despresc · Pull Request #66 · jgm/pandoc-types," *GitHub*. Mar. 2020, Accessed: Apr. 06, 2020. [Online]. Available: https://github.com/jgm/pandoc-types/pull/66.

[77] "How To Build and Deploy a Flask Application Using Docker on Ubuntu 18.04," *DigitalOcean*. Accessed: Apr. 11, 2020. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-build-and-

deploy-a-flask-application-using-docker-on-ubuntu-18-04.

[78] “Dockerize your Python Application,” *Runnable Docker Guides*. Jul. 2016, Accessed: Apr. 11, 2020. [Online]. Available: https://runnable.com/docker/python/dockerize-your-python-application.

[79] C. Chuck, “How to Create a MySql Instance with Docker Compose,” *Medium*. Apr. 2019, Accessed: Apr. 11, 2020. [Online]. Available: https://medium.com/@chrischuck35/how-to-create-a-mysql-instance-with-docker-compose-1598f3cc1bee.

[80] Tom Pollard *et al.*, “Template for writing a PhD thesis in Markdown.” Zenodo, Jul. 2016, doi: 10.5281/zenodo.58490.

[81] “Pgfgantt - Gantt chart package,” *TeX - LaTeX Stack Exchange*. Accessed: Oct. 03, 2019. [Online]. Available: https://tex.stackexchange.com/questions/63877/gantt-chart-package.

[82] D. Cohen and C. Matos, “Third Year Projects – Rules and Guidelines.” Royal Holloway, University of London, 2013.

Glossary

Bluetooth device: In the context this refers exclusively to the Raspberry Pi module that lecturers provision and bring to the lecture for students to authenticate against.

Cryptogram: A challenge that has been signed. This can refer to either the challenge signed by the backend, *or* the challenge signed by the backend and the Bluetooth device.

Class: A single lecture with an immutable start time. This has a course code (to which an arbitrary number of classes can be created). Students and lecturers are allocated against a course arbitrarily. There is no concept of academic or calendar year.

Fingerprint: The browser fingerprint calculated by the frontend JavaScript.

Certificate bundle: The keys and other data generated by the backend for the Bluetooth device to sign data provided by student devices.

MVP: For the purposes of the project, a Minimum Viable Product will be produced. This is enough to prove the concept and related technologies, but is cut down and not production ready.

Appendix

Appendix 0: Python script for base station emulation

```python
#!/usr/bin/env python3

"""
Based on MIDI version by:
    Nick Mooney
    nmooney@cs.washington.edu
    https://github.com/nickmooney/turning-clicker/blob/master/simple_clicker_midi.py
Adapted to remove MIDI logic
"""

import serial
import serial.tools.list_ports
import sys
import argparse
import re
import signal
import csv

parser = argparse.ArgumentParser(description='Base station emulation for Response Card')
parser.add_argument('--verbose', action='store_true',
                    help='Set logging verbosity')
parser.add_argument('--output', action='store',
                    help='Provide output file to be written to (CSV)')
# Constants
INPUT_LENGTH = 60
INCOMING_RE = r"^incoming: (.{12}) --> (.)"
# Global variables
file = None
decoded = []

def decode(line):
    """
    Decodes lines from the clicker base station Arduino to parse addresses and button inputs.
    This function is essentially a direct adaption of line 50 from Nick Mooney's original work
    The regular expression used was written by him
    :param line: The line from the Arduino serial port to decode
    """

    # Handle as UTF-8 and do the Regex match
    line = line.decode("utf-8")
    result = re.match(INCOMING_RE, line)

    if result:
        groups = result.groups()
        data, button = groups
        data_dict = {"address_from": data[:6], "address_to": data[6:12], "button": button}
        return(data_dict)
    else:
        return None

def signal_handler(signal, frame):
    """
    Catches `ctrl + c` and handles gracefully to write CSV to disc
    Based on StackOverflow answer: https://stackoverflow.com/a/57787316
    """
    print("Exiting...")

    if(file):
        writer = csv.DictWriter(file, decoded[0].keys())
        writer.writeheader()
        writer.writerows(decoded)
        file.close()
        print("Written to file successfully")

    sys.exit(0)

def open_file(path):
    return open(path,"w+")

if __name__ == "__main__":
    # SIGINT handler to allow graceful shutdown
    signal.signal(signal.SIGINT, signal_handler)

    args = parser.parse_args()
    ports = serial.tools.list_ports.comports()

    if len(ports) == 0:
        sys.exit("No serial bus devices")

    for n, port in enumerate(ports):
        print("[" + str(n) + "] " + str(port))

    chosen_value = int(input("Enter number of chosen serial device: "))

    port = serial.Serial(ports[chosen_value][0], baudrate = 115200, timeout = .25)
    # If we need to, open a file
    if(args.output):
        try:
            file = open_file(args.output)
        except:
            exit("Path " + str(args.output) + " could not be written to")

    while True:
        if port.inWaiting():
            from_board = port.read(INPUT_LENGTH)
            lines = from_board.split(b"\n")

            for line in lines:
                # Get data with correct charset and then decode lines
                # of useful clicker input
                line_decoded = decode(line)
                if(args.verbose):
                    print(line_decoded)
                if(line_decoded != None):
                    print("Data received")
                    decoded.append(line_decoded)
```

Appendix 1: HTML for Browser Fingerprinting test

```html
<html>

<head>
    <title>Device Fingerprinting</title>
    <!-- Source: https://github.com/Valve/fingerprintjs2-->
    <script type="text/javascript" src="fingerprint2.js"></script>
    <!-- Source: https://github.com/jackspirou/clientjs-->
    <script src="client.min.js"></script>
</head>

<body>
    <h1>Your Fingerprint2 is: <code id="fp1"></code></h1>
    <h1>Your ClientJS fingerprint is: <code id="fp2"></code></h1>
</body>

</html>
```

Appendix 2: JavaScript for Browser Fingerprinting test

```javascript
setTimeout(function() {
    // Fingerprint 2
    var options = {
        excludes: {
            userAgent: true,
            language: true
        }
    }
    // Based off https://github.com/Valve/fingerprintjs2#usage
    Fingerprint2.get(options, function(components) {
        var values = components.map(function(component) {
            return component.value
        })
        var murmur = Fingerprint2.x64hash128(values.join(''), 31)

        document.getElementById("fp1").innerHTML = murmur;
    })

    // ClientJS
    var client = new ClientJS();

    var fingerprint = client.getFingerprint();

    document.getElementById("fp2").innerHTML = fingerprint;

}, 500)
```

Appendix 3: student ID full tag output

```
** TagInfo scan (version 4.24.5) 2019-11-27 17:44:39 **
Report Type: External

-- IC INFO ------------------------------

# IC manufacturer:
NXP Semiconductors

# IC type:
MIFARE Classic EV1 (MF1S50)

-- NDEF ------------------------------

# No NDEF data storage present:
Maximum NDEF storage size after format: 716 bytes

-- EXTRA ------------------------------

# Memory size:
1 kB
* 16 sectors, with 4 blocks per sector
* 64 blocks, with 16 bytes per block

# IC detailed information:
Full product name: MF1S507xX/V1

# Originality check:
Signature verified with NXP public key

-- FULL SCAN ------------------------------

# Technologies supported:
MIFARE Classic compatible
ISO/IEC 14443-3 (Type A) compatible
ISO/IEC 14443-2 (Type A) compatible
```

```
# Android technology information:
Tag description:
* TAG: Tech [android.nfc.tech.NfcA, android.nfc.tech.MifareClassic, android.nfc.tech.NdefFormatable]
* Maximum transceive length: 253 bytes
* Default maximum transceive time-out: 618 ms

# Detailed protocol information:
ID: E2:F8:0C:77
ATQA: 0x0400
SAK: 0x08

# Memory content:
Sector 0 (0x00)
[00] r--  E2 F8 0C 77 61 88 04 00 C8 23 00 20 00 00 00 18 |...wa....#. ....|
[01] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[02] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[03] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key           Factory default key (readable)

Sector 1 (0x01)
[04] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[05] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[06] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[07] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key           Factory default key (readable)

Sector 2 (0x02)
[08] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[09] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[0A] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[0B] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key           Factory default key (readable)

Sector 3 (0x03)
[0C] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[0D] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[0E] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[0F] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key           Factory default key (readable)

Sector 4 (0x04)
[10] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[11] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[12] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[13] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key           Factory default key (readable)

Sector 5 (0x05)
[14] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[15] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[16] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[17] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key           Factory default key (readable)
```

```
Sector 6 (0x06)
[18] ???  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[19] ???  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[1A] ???  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[1B] ???  XX:XX:XX:XX:XX:XX --:--:-- -- XX:XX:XX:XX:XX:XX
          (unknown key)                (unknown key)

Sector 7 (0x07)
[1C] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[1D] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[1E] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[1F] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key          Factory default key (readable)

Sector 8 (0x08)
[20] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[21] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[22] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[23] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key          Factory default key (readable)

Sector 9 (0x09)
[24] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[25] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[26] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[27] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key          Factory default key (readable)

Sector 10 (0x0A)
[28] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[29] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[2A] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[2B] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key          Factory default key (readable)

Sector 11 (0x0B)
[2C] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[2D] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[2E] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[2F] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key          Factory default key (readable)

Sector 12 (0x0C)
[30] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[31] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[32] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[33] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key          Factory default key (readable)

Sector 13 (0x0D)
[34] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[35] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[36] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[37] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
```

```
          Factory default key           Factory default key (readable)

Sector 14 (0x0E)
[38] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[39] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[3A] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[3B] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key           Factory default key (readable)

Sector 15 (0x0F)
[3C] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[3D] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[3E] rwi  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
[3F] wxx  FF:FF:FF:FF:FF:FF FF:07:80 69 FF:FF:FF:FF:FF:FF
          Factory default key           Factory default key (readable)

r/R=read, w/W=write, i/I=increment,
d=decr/transfer/restore, x=r+w, X=R+W
data block: r/w/i/d:key A|B, R/W/I:key B only,
  I/i implies d, *=value block
trailer (order: key A, AC, key B): r/w:key A,
  W:key B, R:key A|B, (r)=readable key
AC: W implies R+r, R implies r

--------------------------------------
```

Appendix 4: Minicom output

```
--- Miniterm on /dev/ttyAMA0  9600,8,N,1 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---

--- pySerial (3.4) - miniterm - help
---
--- Ctrl+]   Exit program
--- Ctrl+T   Menu escape key, followed by:
--- Menu keys:
---    Ctrl+T  Send the menu character itself to remote
---    Ctrl+]  Send the exit character itself to remote
---    Ctrl+I  Show info
---    Ctrl+U  Upload file (prompt will be shown)
---    Ctrl+A  encoding
---    Ctrl+F  edit filters
--- Toggles:
---    Ctrl+R  RTS   Ctrl+D  DTR   Ctrl+B  BREAK
---    Ctrl+E  echo  Ctrl+L  EOL
---
--- Port settings (Ctrl+T followed by the following):
---    p          change port
---    7 8        set data bits
---    N E O S M  change parity (None, Even, Odd, Space, Mark)
---    1 2 3      set stop bits (1, 2, 1.5)
---    b          change baud rate
---    x X        disable/enable software flow control
---    r R        disable/enable hardware flow control
--- Settings: /dev/ttyAMA0  9600,8,N,1
--- RTS: active    DTR: active    BREAK: inactive
--- CTS: active    DSR: inactive  RI: inactive  CD: inactive
--- software flow control: inactive
--- hardware flow control: inactive
--- serial input encoding: UTF-8
--- serial output encoding: UTF-8
--- EOL: CRLF
--- filters: default
--- local echo active ---
AT
ABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABC

--- Settings: /dev/ttyAMA0  9600,8,N,1
--- RTS: active    DTR: active    BREAK: inactive
--- CTS: active    DSR: inactive  RI: inactive  CD: inactive
--- software flow control: inactive
--- hardware flow control: inactive
--- serial input encoding: UTF-8
--- serial output encoding: UTF-8
--- EOL: CRLF
--- filters: default

--- Settings: /dev/ttyAMA0  9600,8,N,1
--- RTS: active    DTR: active    BREAK: inactive
--- CTS: active    DSR: inactive  RI: inactive  CD: inactive
--- software flow control: inactive
--- hardware flow control: inactive
--- serial input encoding: UTF-8
--- serial output encoding: UTF-8
--- EOL: CRLF
--- filters: default
```

Appendix 5: HM-10 Test Script

```python
import serial, time

ser = serial.Serial(
        port='/dev/ttyUSB0',
        baudrate=9600,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=1
)

print("Serial is open: " + str(ser.isOpen()))

ser.write(b'ABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABC'
print(str(ser.read(100)))

time.sleep(2)
```

```
ser.write(b'AT')
print(str(ser.read(100)))

ser.write(b'AT+ADDR?')
print(str(ser.read(100)))

ser.close()
```

Appendix 6: Web Bluetooth API test

```
<html>

<head>
    <title>Web Bluetooth</title>
</head>

<body>
    <button id="button" onclick="makeBLE()" type="button">Make BLE</button>
</body>

<script>
    function makeBLE() {
        console.log("Connecting");
        navigator.bluetooth.requestDevice({
                filters: [{
                    services: [0x484D536F6674]
                }]
            })
            .then(device => {
                // Human-readable name of the device.
                console.log(device.name);

                // Attempts to connect to remote GATT Server.
                return device.gatt.connect();
            })
            .catch(error => {
                console.log(error);
            });
    };
</script>

</html>
```

Appendix 7: Plan Risk Assessment

Device communication

- **Likelihood:** Possible
- **Impact:** Medium

Previously in the project proposal it was stated that the Web Bluetooth API would be used. I have decided to carry out additional research into the various communications technologies available to determine the most appropriate for this project. It is possible that there will be no good solution for direct device to device communication (this is intended to provide the “presence” factor of the authentication) in which case a more traditional challenge-response protocol with rotating codes displayed to students, may be appropriate. This reduces security but the ultimate aim is for this to be practical to implement. It is not envisioned this will be a “showstopper”, but it would require additional research and analysis to determine an appropriate way forward.

Analysis of clicker system

- **Likelihood:** Unlikely
- **Impact:** Medium

This relies on the research of previous (cited) authors and it is not feasible for the investigation they carried out to be completed for this project. The intention of this section is to demonstrate a live proof of concept using their discoveries applied to the specific Royal Holloway situation - this will require developing software and a test environment. It is possible that Turning Technologies have changed the way their devices work - in this case reasonable efforts can be made to reverse engineer the devices further and research more up to date analysis. At the point firmware is needing to be dumped the benefits of including this in the report start to outweigh the time needed to complete this section, and it detracts from the rest of the project. This section will produce useful learnings either way, but as with all penetration testing, the exact outcome cannot be predicted in advance.

Frontloading functional design

- **Likelihood:** Likely
- **Impact:**: Low

The milestones above call for the functional design to be broadly complete by the end of the first term, based on the research and proof of concept programs already developed. It is envisioned that much of the code developed during that term will be reworked and make it’s way into the final release. This creates the danger that when integrating the system and building the wireframes out as a frontend panels that the architecture may need to change as the result of either changed requirements or oversights in the design process. The only mitigation to this is to ensure the code is well designed and documented, such that it can be modified as required to adapt to changing requirements. The design process should also be robust and based on the research, but it is not reasonable to expect no changes to be made further down the line.

Appendix 8: User Stories

Student

1) “As a student I want to register my attendance so I can prove I was at a lecture”
2) “As a student I want to register my attendance without distraction from the lecture content”
3) “As a student I want to be able to know my attendance has been counted”
4) “As a student I want to be able to see my attendance history”
5) “As a student I want to register with the minimum of additional hassle”
6) “As a student I don’t want to have to remember extra items to register”

Lecturer

1) “As a lecturer I want to not have to think about lecture registration”
2) “As a lecturer I want to have the minimum of distractions and the minimum of fuss, in registration”
3) “As a lecturer I don’t want to be involved in the registration process”

Administrator

1) “As an administrator I want to have all the registration data in a machine readable format”
2) “As an administrator I want to be able to easily identify students who have not reached minimum attendance levels”
3) “As an administrator I want to easily identify cases of fraud or forgery”
4) “As an administrator I want to generate reports on attendance and export raw data”

Appendix 9: Test framework output

```
================================================= test session starts =================================
platform linux -- Python 3.7.5, pytest-5.1.1, py-1.8.0, pluggy-0.12.0
rootdir: /home/crablab/Documents/RHUL/year_3/FullUnit_1920_HughWells/backend
plugins: cov-2.8.1, flask-0.15.1
collected 36 items

app/tests/allocation_test.py ........
app/tests/allocations_test.py ..
app/tests/lecture_test.py ......
app/tests/lectures_test.py ...
app/tests/user_test.py ..............
app/tests/users_test.py ..

================================================= 36 passed in 2.01s =================================
```

Appendix 10: Project Diary

October 21, 2019

Monday:

- Have spent the day debugging issues with serial communication
- Have written a small Python script to try sending data which isn't working
- Discovered that I'm not actually broadcasting the hardcoded messages I thought I was (or at least they aren't valid). I'm trying to determine why this is...

October 28, 2019

- Finished writing up report on clicker emulation for now
- Additionally, gave up on getting the final pieces of clicker emulation to work for the moment: I will come back to it over a period of time once I've had a chance to consult the internet and Nuno some more, to work out what is going wrong!
- Begun researching communication methods for my proposed solution and preemptively ordered a HM-10 Bluetooth 4 module for further investigation

November 4, 2019

- Wrote some more about clicker communication
- Got a Raspberry Pi working over Serial with Ethernet passthrough
- Determined that it's not possible to connect to a Pi over Serial and utilize UART on the GPIO. Although the Broadcom chip support this, it is not connected in a way that allows both these Serial connections at once unless you use the Compute Module (the professional version of the Pi)
- Going to look into other options here as I want to avoid using an Arduino as I would like the networking capabilities of the Pi and the additional processing capacity. It also allows me to work with more standard cryptography libraries.

November 4, 2019

- Switched from a Pi Zero to a Pi 2 and now connecting via SSH with ethernet passthrough
- UART is now working, testing to try and get communications working with the HM-10

November 5, 2019

- I have a serial connection open, but I am not receiving data back. The board does have UART flow control pins and although I have tried the hacks (hold the enable pin low), I am still not getting any data back. Enabling hardware flow control on the Pi is possible, but involved. I have been testing with minicom and all of the Google solutions are still not enabling it.

November 13, 2019

Yesterday:

- Worked on more issues with the UART. I think I have found a solution but it seems it requires setting a config parameter on the board itself, so I have had to purchase some additional equipment to do this.

Today:

- Researched and wrote a PoC for fingerprinting web browsers and devices
- Researched PKI and RSA vs ECDSA algorithms. Wrote a quick Adaptor class with tests to play around with a library for this. To be extended.

Current blocker is getting the Bluetooth working. Once that is done it should be fairly easy to send data end to end (he says. . .)

November 14, 2019

- CeDAS workshop on report writing with very useful context and helpful information about how to structure the report. Have begun laying out interim report.
- Meeting with supervisor. Discussed progress and where to concentrate work from now on. Discussed how reports are not final and feedback is really useful on work that isn't quite complete. Structure of interim report clarified.
- Started writing up investigations into browser fingerprinting. I shall probably provide an extra POC program here for illustrative purposes.

November 18, 2019

- Discovered an article that suggested this might well be a logic level issue on the RX pin (3.3v vs 5v) and that there are two different baud rates depending on software.
- Took notes on BLE specification stuff for a report on communication options
- Discussed project in general with Dave – the various reports and the structure the project should take, plus the specific Bluetooth issues. Dave has suggested that as per another article there could be an issue with AT commands missing a line feed/carriage return, causing the null byte issue I'm seeing

November 20, 2019

Yesterday:

- Chatted with supervisor about clicker report. Feedback about general tone and structure of report.
- Discussed general project outline and progress to date
- Feedback on current report proposals. Agreed to add an additional report on general project ideas and outline, as without it the context of the project is somewhat confusing for an outsider.
- Communication report to be written, cryptography report to be delayed in light on the additional outline report and browser fingerprinting work.

Today:

- Implemented changes suggested in clicker report[4]
- Working on outline report

[4]https://github.com/RHUL-CS-Projects/FullUnit_1920_HughWells/pull/5

November 28, 2019

Earlier this week:

- Finished writing up the Outline report covering the basic system design and user stories
- Meeting with Supervisor to discuss this report, feedback and changes
- Significant research and compilation of notes on BLE specifications

Yesterday:

- Progress on Communications report theory. MIFARE attack covered and Bluetooth section complete up to part way through security features

Today:

- Finally managed to get HM-10 devices working properly. This has been delayed by waiting for hardware to arrive in dribs and drabs. An HM-10 from a different supplier and manufacturer arrived and, with no changed settings, has "magically" started working.
- I can now send data from a phone, through the HM-10 to a serial console and back again. My script to test AT commands works and various notifications are logged to the serial port when device changes take place (eg. disconnection). This is huge progress and I shall now be able to write a driver for the HM-10.
- Started work on a Web Bluetooth implementation. Have discovered that browser support is *shocking.* I currently am running experimental versions of Chromium (with flags enabled) and bluez (flags enabled) and am still encountering errors. I shall need to look further into this but it may prove unfortunatly rather fatal if I cannot find some combination of hardware that has reasonable support. Will discuss with supervisor.

January 20, 2020

Last week:

- Wrote design report covering the proposed system design including system UML and software architecture decisions
- Started implementing webservice codebase

January 25, 2020

- Started using GitHub Kaban board[5]
- Setup database server (locally) and ORM
- Added design patterns to the design
- Started the final report

February 3, 2020

Last week:

- Wrote the professional issues section of the report
- Designed database schema with constraints and dependencies

February 4, 2020

Today:

- Making my Flask application more scalable and better structured – Started using basic Blueprints and restructured the application – Read/learnt about building basic applications in this format with SQLAlchemy and various Flask packages to help abstract functionality – Looked at unit and functional testing – Looked into scaling issues and fan-out – how do you handle databases at scale?

[5]https://github.com/RHUL-CS-Projects/FullUnit_1920_HughWells/projects/1

TODO:

- Implement full Flask factory
- Add the new ORM structure and a full class with unit tests (probably a user login page)

February 9, 2020

- Transferred to using Blueprints and got it working!
- Using Jinja, Boostrap and What the Form have made login and signup page templates, and started on other application modules

February 17, 2020

Added some more pages and tried to get unit testing working. Refactored the application again to make it more scaleable and get unit testing working with the database. Focus is now going to be on getting some working code, with more basic unit testing.

February 18, 2020

Got the login and signup pages working. This require implementing all the logic and debugging some painful issues with flask_login.

March 3, 2020

Last week:

- Implemented users model with associated testing

This week so far:

- Implemented lectures and allocations to lectures
- Administrators page
- Hookins to allow allocations from the UI
- Fixed some bugs/issues

March 10, 2020

Today:

- More report writing, including the manual
- Tried to Dockerise Arduino build environments – very fragile and didn't work
- Looking further into WebUSB, Arduino Micro ordered

March 18, 2020

Worked on adding more functionality for administrators and students. Merged today the functionality to allow administrators to add courses and lectures, plus students to view upcoming lectures and their next lecture.

April 12, 2020

Finished writing up the project. Merged changes with Web USB experiments and the Dockerised Flask application

Appendix 11: User Guide

This document provides a set of instructions for setting up, running and using the various software deliverables provided.

Some hardware is required. It will be listed in the respective sections and provisions have been made with Dave Cohen (d.cohen@rhul.ac.uk) for it to be available over the summer.

Main software deliverable

The main deliverable is a Python Flask application. This is provided with Dockerfiles to enable it to be run with ease on your machine.

Requirements:

- Docker installation: https://docs.docker.com/get-docker/
- docker-compose: https://docs.docker.com/compose/install/

1) In the project directory, go to the `backend` directory.
2) Execute `docker-compose up` and go and make a cup of tea!
3) Visit http://localhost:5000 to get the login page

To terminate `ctrl+c` and the containers will gracefully stop. The database is persisted.

Using the website

To create a new user you visit `/signup`. It is recommended to create a student and administrator account.

You can then login as an administrator and create a new course with first lecture in the future. You can add further lectures if required.

It is then possible on `/administrator` to assign students to a course.

Logging in as a student, you see all upcoming lectures for all courses, and the next lecture.

Running tests

As explained in the report, these are integration tests so in order to run them a local environment and database is required.

One way to achieve this is to run just the MySQL container in Docker and then connect to that from your locally running tests.

Requirements:

- Python3
- Local installation of `requirements.txt` with Pip
- Installation of `pytest` using Pip

1) Modify the connection string `backend/app/libraries/database.py` to refer to your instance of MySQL
2) Execute `pytest` in `backend/app/tests`

Alternatively, you can run both containers as described above and then enter the `web` container, kill the running Python process, install `pytest` and run the tests on that container.

Common Issues

If you encounter any database connection issues then it's likely you just need to wait a little longer for the database to start. It is restarted several times when the container is being created and the website will be available long before it can connect to the database container.

`clicker_emulator`

This software is made up of two parts:

- An Arduino sketch
- A Python console to communicate with the Arduino

Requirements:

- Python3
- Arduino IDE
- POSIX-compliant Operating System

Hardware

Any Arduino should be sufficient to run this code. The project was carried out on a Nano328, however. A nRF24E1 is required and the pinout to connect it to the Arduino is in {**???**}.

Installation: Arduino

The setup of the software environment on the computer is relatively simple and is based around the Arduino IDE which is available from: https://www.arduino.cc/en/main/software. It appears there is now an online version, but this has not been tested with this project.

Once the Arduino IDE is installed you should try flashing the Arduino with the example "blink" code, available from File → Examples → Basic.

Assuming you have installed the IDE correctly and have specified the serial port and Arduino type, a blinking status LED will be present.

You can also compile and upload code from within other IDEs using plugins. VSCode has various plugins for this, setup of which is beyond the scope of this document. However, unintuitively, it is important that you set up a VSCode Workspace so the Arduino Sketch file (containing running preferences such as the serial port, baud rate etc.) can be created and saved. Otherwise, you cannot upload any code.

To run the project code, two libraries are required:

- RF24 (a library to handle communication with the nRF24E1)
- FastCRC (to calculate the CRC checksums)

For the former, Sparkfun have published a guide on installation of the RF24 library into the correct folder in your filesystem: https://learn.sparkfun.com/tutorials/nrf24l01-transceiver-hookup-guide/arduino-code and you can download the RF24 library here: https://github.com/nRF24/RF24

The FastCRC library is installed in the same directory, and is downloadable here: https://github.com/FrankBoesing/FastCRC

You should then be able to verify and upload the project code! If you get library errors, ensure the libraries have been imported correctly and the IDE shows them in the GUI.

The baud rate used in the project is 115200 - without that set correctly it will not be possible to communicate with the Arduino via the IDE's serial console.

Frequent Issues

Two issues encountered and which took significant effort to identify the root cause of are included with some basic debugging.

`fatal error: <library>: No such file or directory`

Check your libraries folder! The error will tell you which library is causing issues and it's important to note that each library should be in it's own folder, named identically to the `.h` file a directory down. Other subdirectories and files will be included.

The expected structure is:

```
- FastCRC
    - FastCRC.h
    - ...
- RF24
```

```
    - RF24.h
    - ...
```

In the case that other libraries are missing, you should be able to install them in a similar fashion.

```
avrdude: stk500_getsync() attempt 1 of 10: not in sync: resp=0x00
```

This error occurs when either:

- The Arduino is disconnected or cannot be connected to: check the USB
- The wrong Arduino type is selected: check the board configuration and the type of CPU used

```
Cannot upload: device/resource busy
```

This occurs when the serial connection to the serial port is already in use. Ensure you don't have any serial consoles open, or any of the Python scripts running. If that doesn't work, you may be specifying the wrong serial port.

It is possible to identify a process using a serial port with `lsof` and then terminate this process using `pkill`.

Installation: Python

The Python script is designed to work in Python 3, and not 2.7.

The only library required and possibly not installed is PySerial, which is installed with `pip3 install pyserial`.

When running the script, if any issues are encountered with packages then installing these with Pip should resolve them.

Running

The Arduino requires nothing except the presence of a power source to run. The code is in a constant loop and to reset the device (on the Nano) there is a button on the top. When the serial bus is active a red LED will flash.

Execute the Python script in your terminal:

```
python3 cli.py
```

And when prompted, enter the numeral of the Arduino:

```
[0] /dev/ttyUSB0 - USB2.0-Serial
[1] /dev/ttyACM0 - Sierra Wireless EM7345 4G LTE - Sierra Wireless EM7345 4G LTE
Enter number of chosen serial device: 0
```

(It can be helpful to run the script without the Arduino connected, terminate with `ctrl + c` and then try again with the device connected, to deduce which device the Arduino is)

When using a Turning Point Clicker in the vicinity on the default channel (41), you will then see log lines appear.

More options are described by running `python3 cli.py --help`

`device_fingerprinting`

This is a basic demonstration page to show two different libraries used for browser fingerprinting.

Requirements:

- Modern (eg. latest stable version) browser

Running

Simply open the `index.html` file in your web browser. Two hashes which serve as the fingerprint will be displayed.

You can see how well the libraries work by opening the page across different tabs, in "incognito" modes and in other browsers.

`crytographic_signing`

This is a simple wrapper (an adaptor pattern) of a Ecliptic Curve signature algorithm.

Requirements: - Python 3

Installation

The only dependency is on the ECDSA library, which can be installed with Pip `pip3 install ecdsa`.

If you wish to run the tests you will need Pytest: `pip3 install pytest`.

Running

As this is intended as an adaptor, it is structured as a library. You can execute the tests by running `pytest` in the directory.

`hm10-investigations`

Originally, the HM-10 was intended to be connected to a Raspberry Pi but using a serial (RS232 TTL) to USB converter, you can connect it to a computer.

1) Install Python3 and pip (the package manager) for your operating system: `sudo apt install python3 python3-pip`
2) Use pip to install the `pyserial` library: `pip3 install pyserial`
3) Modify the `initial_test.py` script to use whichever port you have the HM-10 connected to (identified with `lsusb`)
4) Execute the script with `python3 initial_test.py`

You can also connect directly to the HM-10's port with `screen`, `minicom` or any other similar utility.

Using one of the BLE apps described in the report, you can then connect to the BLE device and observe sending data to the BLE device and receiving it on the serial port.