Internet of Things Systems Security
System Specification
UFCF8P-15-M

Luke Murray (19041865)
and
Jacob Williams (15008632)

Project Source Folder:
https://gitlab.uwe.ac.uk/jj6-williams/iots_task3/tree/master/source

Project Documentation Folder:
https://gitlab.uwe.ac.uk/jj6-williams/iots_task3/tree/master/documentation

Table of Contents

Hardware and Functionality

The specification for this coursework proposes the possibility of using different devices from the previously used Micro-bit, such as the Raspberry Pi or an Arduino. However, since thus far the Micro-bit has been utilized primarily it would make sense to continue using the Micro-bit. This also has the added bonus of experience working with an unconventional IoT device that has little documentation, giving a relatively fresh experience of IoT programming.

The second consideration will be which communication method to use with the Micro-bits. The choices being Radio or Bluetooth Low Energy. If using the Radio functionality it is possible to continue using the already familiar DAL run-time, however if one wishes to use BLE one would have to switch to using ARM Mbed. Both functionalities are provided by Lancaster University.

The Radio functionality is extremely simplistic, it works as one might expect. A device acts as the transmitter, another device idles using the DAL listen function to receive the communication and then one programs what happens as a result of what has been received. Lancaster University say that the Radio functionality is designed with “privacy in mind”, given that “there is nothing inherent to the protocol which can be used to identify you or your micro-bit”. However privacy doesn’t necessarily ensure security, given the open nature of the radio communication an attacker could technically intercept communications and even masquerade as either the sender or receiver if they so desired.

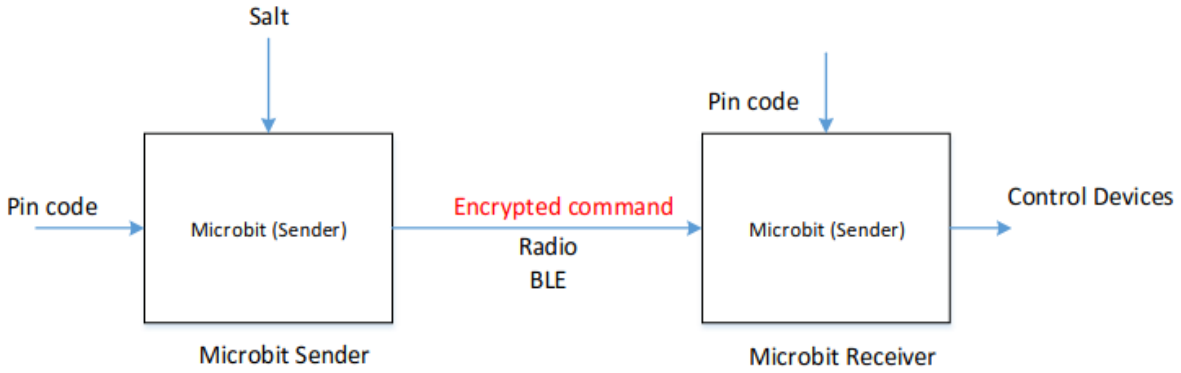
The Bluetooth functionality provides various security functions, including being able to pair with passkeys and including different protected communications. It also has white listing and the links between devices are encrypted by default. The draw back of BLE is that one would have to change the run-time and also develop for multiple different devices. As a result this method would be considered more difficult in the context of this project.

The method deigned to be used was Radio, its functionality sounds interesting and finding ways to incorporate security into a messaging functionality that is no way by default secure is an interesting prospect.

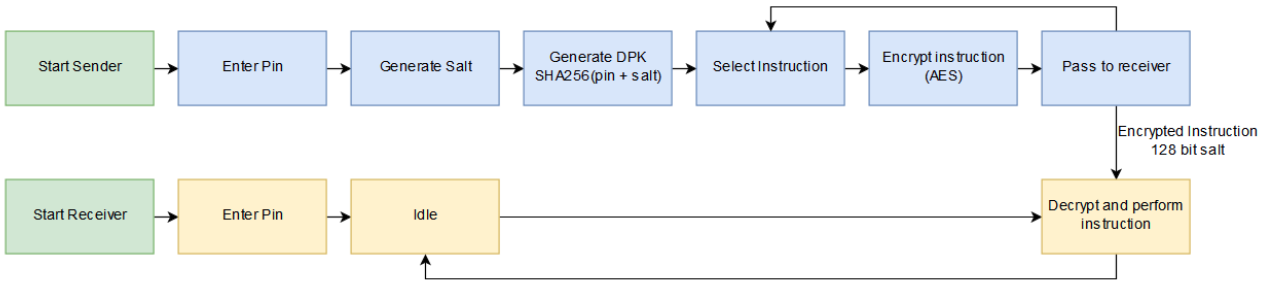
Design

Layman's Run Time

There is a example runtime laid out in the coursework specification, this being:



The runtime being that the microbits are started, pins are entered on both microbits. The sender then generates a salt, which is combines with the pin code and encrypts in SHA256, giving us a 64 byte data protection key to use with the AES encryption later.



Above is a flow representation of the two roles the microbit can operate under, in theory one implementation can be created to perform both roles, where one simply chooses the desired role at the beginning of the runtime. The sender should be programmed to work like a hub, where it distributes commands to the recievers within range of it. The reciever should be designed in such a way that the number of recievers can be dynamic, anything running the reciever side of the program in range of the corresponding sender with the same pin should be able to receive the instructions.

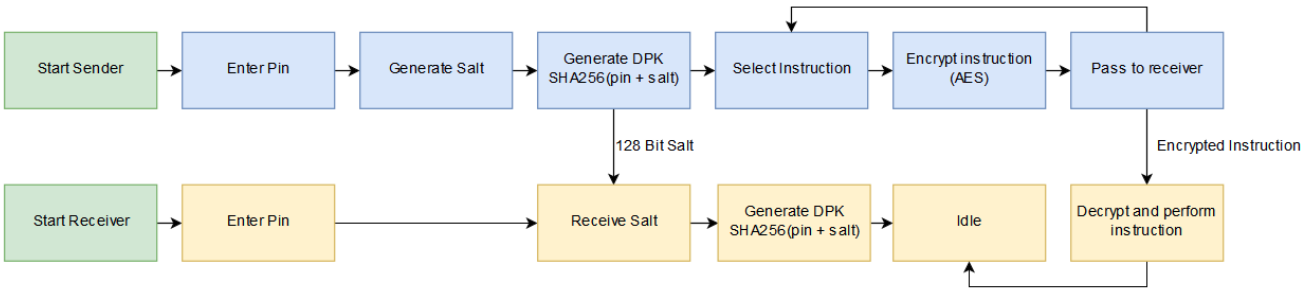
There is however a key flaw with the example runtime that stops it from being the basis of this project. The design implies that the salt is passed every time the sender sends an instruction which seems like a huge security flaw, as possible intruders intercepting the messages would see the repitition and possibly be able to breach the method of passing instructions.

Counteracting this would be a big undertaking, but an idea of how to reduce the potential entry of attack has been formulated, nicknamed “saltshaking”.

Saltshaking

The concept of handshaking is an old and established protocol for linking two devices, effectively working by the sender sending an identifier, the receiver returning a confirmation of the identifier, and then the sender sending a confirmation of receiving the confirmation. This works to establish the connection and not require the passing of any extra identifiers after handshaking.

The Saltshaking implementation works under a similar basis. The sender distributes the Salt to the receivers before sending any instructions, so that when it comes to actually sending instructions there is no need to pass the salt multiple times.



In theory this implementation will reduce the potential of intercepting the salt and therefore potentially being able to breach the communication channel of the two Micro-bits, as all messages occurring after the Saltshake will be encrypted using the resultant DPK.

This method doesn't counteract other possible security risks occurring from the factor of using the Micro-bit radio communication, such as intruders being able to masquerade as sender. But this method does reduce the potential effectiveness of an interception attack given that in essence the attacker has one chance to do it.

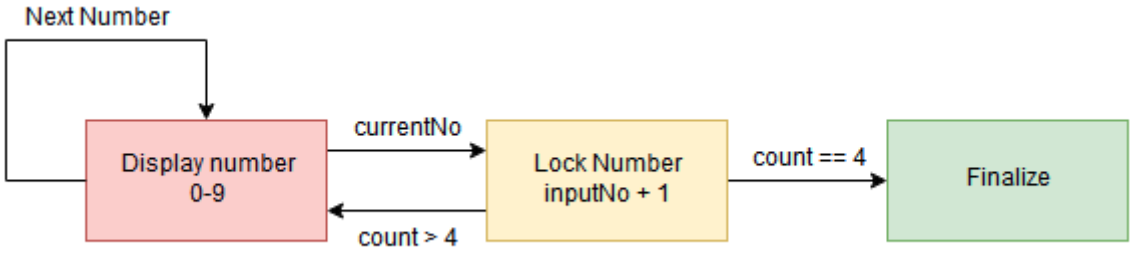
This method also creates another issue, the Saltshake can only in theory occur once, else it defeats its entire purpose. Because of this, if one unit in the cluster of devices loses its DPK (losing power, being manually reset) then the entire cluster will need to be re-initiated.

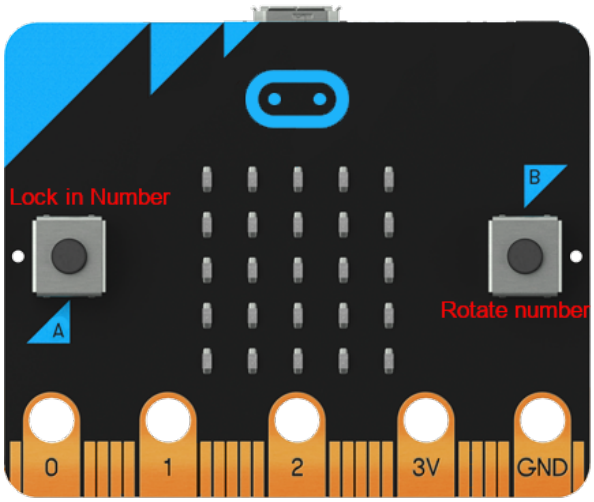
Pin Code Entry

The initial problem of pin code entry boiled down to hardware, the external functionality of the Micro-bit. The Micro-bit has the ability to be used with other external peripherals thanks to its 40 general purpose input output headers (GPIO headers) and the breadboard supplied with it. It would be entirely possible to purchase an external membrane keypad and use a lookup matrix to get the desired pin code. Whilst this would certainly be effective, it was deemed necessary to explore methods of entering the pin code without resorting to extra purchases, and as such experimenting with the functionality the Micro-bit inherently had began.

Rotary Dialer

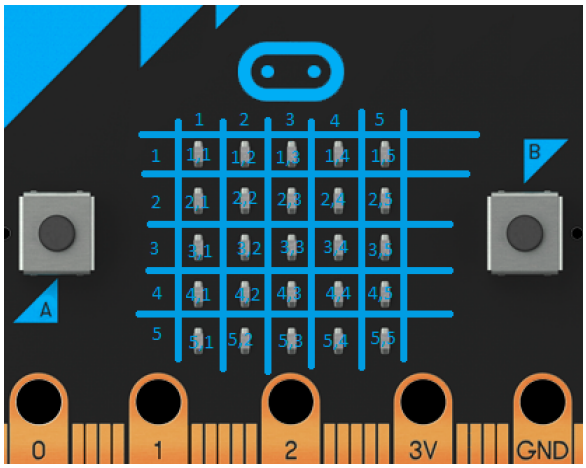
A now niche relic of the past, the rotary dialer is a way of inputting phone numbers into your handset by rotating the central wheel to the desired number and waiting until it reset before entering the next. This implementation idea was spurred by this concept, what method can you use to enter a pin code when you have two buttons by default? Simply, one “rotates” the numerical value and the other locks it in. The reason this is reminiscent of the rotary dialer is that they both utilize the functionality that to access the higher numbers you have to bypass the lower ones first. It works on a ‘0-9’ basis, once it reaches 9, pressing next number reverts it to 0.

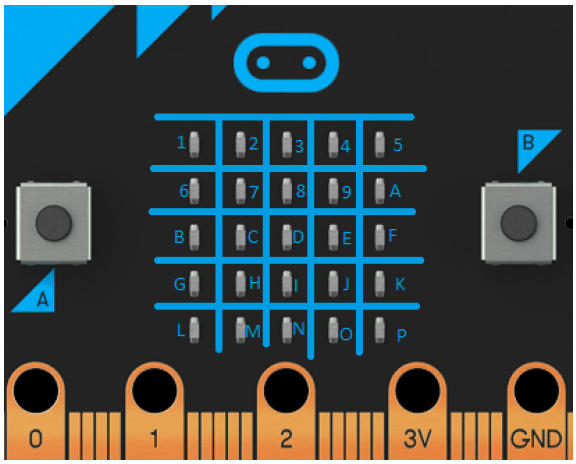




Pattern Pin (Conceptual)

The crux of this method is utilizing the visualization of the Micro-bit LED board to interpret a pin code for the creation of the DPK. This method is only works due to the fact that the pin code entered doesn't have to be limited to simply numbers like the average pin code might be, so as a result we can map particular values to each of the LED positions on its board, all the user would have to do is pick a number of different positions on the board.





Each LED has a particular value in the run-time, a coordinate positional value ranging from 1,1 to 1,5. Each of these 25 positions will have to be assigned a fixed constant value in order to sync the multiple Micro-bits.

The amount of entries in the pin is arbitrary, the default would be 4 simply because this is the usual index for a pin number, but in theory it could be more. The upper bound in theory would be under half, or 12 entries, anymore than this an you end up with a predictable pattern, the pattern should also under no circumstances be able to be the entire board or none of the board.

This method would in theory create a secure method of pin entry, as for all intents and purposes even the person entering the pattern doesn't know the precise pin values, they merely know the pattern that represents them, meaning that once the pin is utilized in the SHA256 DPK is will have be obfuscated from potential attackers twice.

Command Definition

In theory the commands should demonstrate different properties of the Micro:bit and therefore should be fundamentally different to each other.

The Micro:bit kit provided supplies the parts to wire up a fan header that can be toggled via outputting through the pin that it is wired up to, the Lancaster University Micro:bit documentation supplies some instruction on how to do this. This would be the primary command, since it displays the implementations ability to control systems outside of itself via its breadboard. Concerning the breadboard, this may have to be the only command that utilizes an output to it, as we don't want to overcrowd the breadboard.

The other functionalities activated by commands will be Micro:bit based, all of these are also documented by the Lancaster University. The second command being a simple pattern output, if the command is received the Micro:bit will display on its LED screen a certain pattern.

The third will make use of the Micro:bit Accelerometer. The Accelerometer is onboard the Micro:bit linked to the i2c bus which reads the data from it. The Accelerometer can detect changes in acceleration as well as changes in orientation. In a practical implementation the Accelerometer could be used to detect when there is a need to deploy airbags, or when the device connected is in free fall. In this implementation we will simply utilize the LED screen to visualize changes in orientation detected by the Accelerometer.

Then the fourth command will use the Micro:bits onboard magnetometer, this is also linked to the i2c bus allowing us to read data from it. The magnetometer allows us to process information on the surrounding magnetic field in order to get an idea of where the magnetic north is. The magnetometer also makes use of the Accelerometer in order to provide an accurate representation of the four cardinal directions. It does this by utilizing the Accelerometer to filter out the vertical z axis, which the Micro:bit may count as north or south depending on orientation.

These four commands make use of various parts that are either supplied with or on board the Micro:bit, meaning that the implementation will display a variety of functionalities that could be performed. Given more time, there may be a possibility of using commands that somehow interact with each other, but for the moment simply displaying different functionalities activated via device communication will suffice.

DPK Generation

Generating the DPK makes use of the pin code whose creation is detailed in previous sections.

Salt Generation

The salt generation is simplistic yet effective. A previous task involved to utilization of a Random Number Generator, the goal of the task to implement the ten tests to verify the random nature of the values created. This random number generator paired with a lookup table containing characters deemed legal for use in salting would effectively create for us a random salt. Our definition of legal characters is broad, and mostly excludes special characters and punctuation characters, leaving the available character list 62 long consisting of “A-Z, a-z, 0-9”.

The random number generator itself utilizes the stdlib rand() function, which has often been criticized for its quality compared to heavier cryptographically secure random number generators, but we believe it more than effectively performs the task it is needed to do in this implementation. Furthermore, the pairing of the salt with SHA256 hashing increases the effectiveness of masking the pin and creating a unique key as even a change to a single bit can change the format of an entire SHA256 string, meaning that even if the random number generator only generates two different characters one time from the last, it would still be enough to alter the entire layout of the SHA256 string.

The salt should be 128 bits long, 64 bytes, 32 characters.

Hashing

The chosen hashing algorithm for this was SHA256, a lightweight and effective hashing algorithm. There is much debate as to the security of SHA256, whilst it hasn’t been cracked its nature as a lightweight and easy to compute hashing function leaves it open and vulnerable to brute force attacks, hence why it is often advised when using SHA256 to also utilize a hashing function, as adding a component of randomness to the hashing can create a very difficult to force hash. SHA256 provides good defense against collision attacks, as it is nearly impossible to find two exactly similar SHA256 hashes, a hash can be made entirely different by changing a singular bit, as proven by an bug encountered and detailed in the next section. SHA256 is however considered to be poor at defending against length extension attacks. The output of SHA256 is a string of characters 256 bits / 32 Bytes / 64 characters long, which conveniently is the key size for AES256 encryption, it would also be possible to use sub-strings of the DPK for lower forms of AES encryption, but given the convenient interplay

between the two functions it would make sense to utilize it to its fullest degree.

SHA256 Variable Encoding Bug

An interesting bug was encountered during the creation of the DPK functionality. On the sender side, the pin was entered, salt generated, and the DPK sucessfully created by hashing the two in SHA256. On the receiver side the pin was entered, the salt was received, the DPK was generated with SHA256. However, the two hashes were different. The difference was not minor either, such a minor difference could be put down to the radio transmission happening too quickly and losing characters during the process, but the hashes were entirely different. Numerous tests were conducted, the SHA256 function was being passed the exact same string of pin and salt, but the two exactly the same inputs were producing two entirely different outputs.

The cause was a fairly strange one, simply the receiver had to utilize a transition variable, a ‘temp’ if you will. This transition variable was the Microbit uBit ‘managedString’, which is used to store the incoming radio transmission for processing, the value of this transition variable was then stored in the array for the salt on the receiving end. For some reason manageString changed the character value functionally but not fundamentally, a change at the bit level that was inconsequential at any other level. When comparing two characters to each other, one c char and the other uBit ManagedString it would return that they are the same character, but then when hashing them the two would return different hash values.

The first solution was to utilize the ManagedString in the creation of the salt on the sender side, which worked, the two salts were now using the exact same encoding and therefore would produce the same hash output. However this was deemed not to be really solving the issue, but rather expanding it to the point where it was no longer affecting this particular functionality, essentially turning a bug into a feature. What concreted the need for another solution was the possible impact this encoding error could have on the future AES implementation. Instead the solution found was to interpret the received character before storing it in the array using a lookup table of all the characters, this solution was derived from the fact mentioned earlier that when comparing the two characters of the different types they would still be deemed that same character. Using this method, the two hashes now match.

Structure

Name	Length	Example
Pin	16 bits / 2 Bytes / 4 Characters	1111
Salt	128 bits / 16 Bytes / 32 Characters	222222222222222222222222222222
PinSalt	144 Bits / 18 Bytes / 36 Characters	111122222222222222222222222222222222
SHA256 Hash	256 Bits / 32 Bytes / 64 Characters	dd9daf5d2d835295ded2a43b27c7e78a0bea5b208190757cf3b78dcc51d74a39

Command Transmission

AES

The desired from of command transmission is to utilize AES and use the DPK as the key. In order to sync the Micro-bits proper, we will have to use ECB mode due to the fact it doesn't possess the initialization vector of the CBC, CTR, etc. Attempting to utilize the AES formats that require the initialization vector would create a great deal of complication in the setup of the Micro-bits, and would require much debate as to what the best method of establishing the initialization vector would be. The most simple way would be to hard code it, which is why this mode would be less desirable, as hard-coding the IV may well make the system less secure.

Hence the utilization of ECB mode, which then can be used in three forms itself: AES 128, 196, 256. These being the relational key sizes of each of the implementations. In theory the most effective and convenient form to use would be 256, as the DPK we generate is 64 characters or 256 bits. The other forms may be easy enough to implement and would simply involve creating sub-strings of the DPK to use as a key rather than using the entire DPK.

The use of AES in this implementation may actually be excessive, as due to the nature of the Micro-bit Radio communication method it doesn't actually provide any security, it merely provides privacy in the form of masked intentions. Would be attackers could quite easily intercept the AES transmission, and re-transmit it, there would be no need for them to actually break the cipher is their only intention is to cause havoc. The developers at Lancaster University (2016) say it themselves, "there is nothing inherent in this protocol which can be used to identify you or your microbit.", this meaning there is no way to discern the person transmitting the command when using the radio method.

Numerous ideas were considered, signing the encrypted command in some form; signing the transmission of the AES blocks. But none of these provide any additional security, none of them stop the potential of replay attacks, they do however remove the privacy that the radio method provides. Essentially, by attempting to make the communication more secure in radio communication, you actually remove one factor of its security, its privacy.

This would mean the primary use of AES in this context would be obfuscation, hiding ones origins, destination, and intentions. Considering this, one would question the utilization of AES, why bother to utilize an elaborate and more weighty form of encryption if in essence it provides no more security than other methods?

SHA256 Hash Messaging

This is the alternate form of private messaging formulated, it is lighter weight than using AES encryption but provides nearly the same level of protection.

It works by assigning unique pins to the commands and combining them with the generated DPK and hashing them in SHA256, creating a completely unique identifier value for the commands. This means the commands are obfuscated once, and the base pin is obfuscated twice.

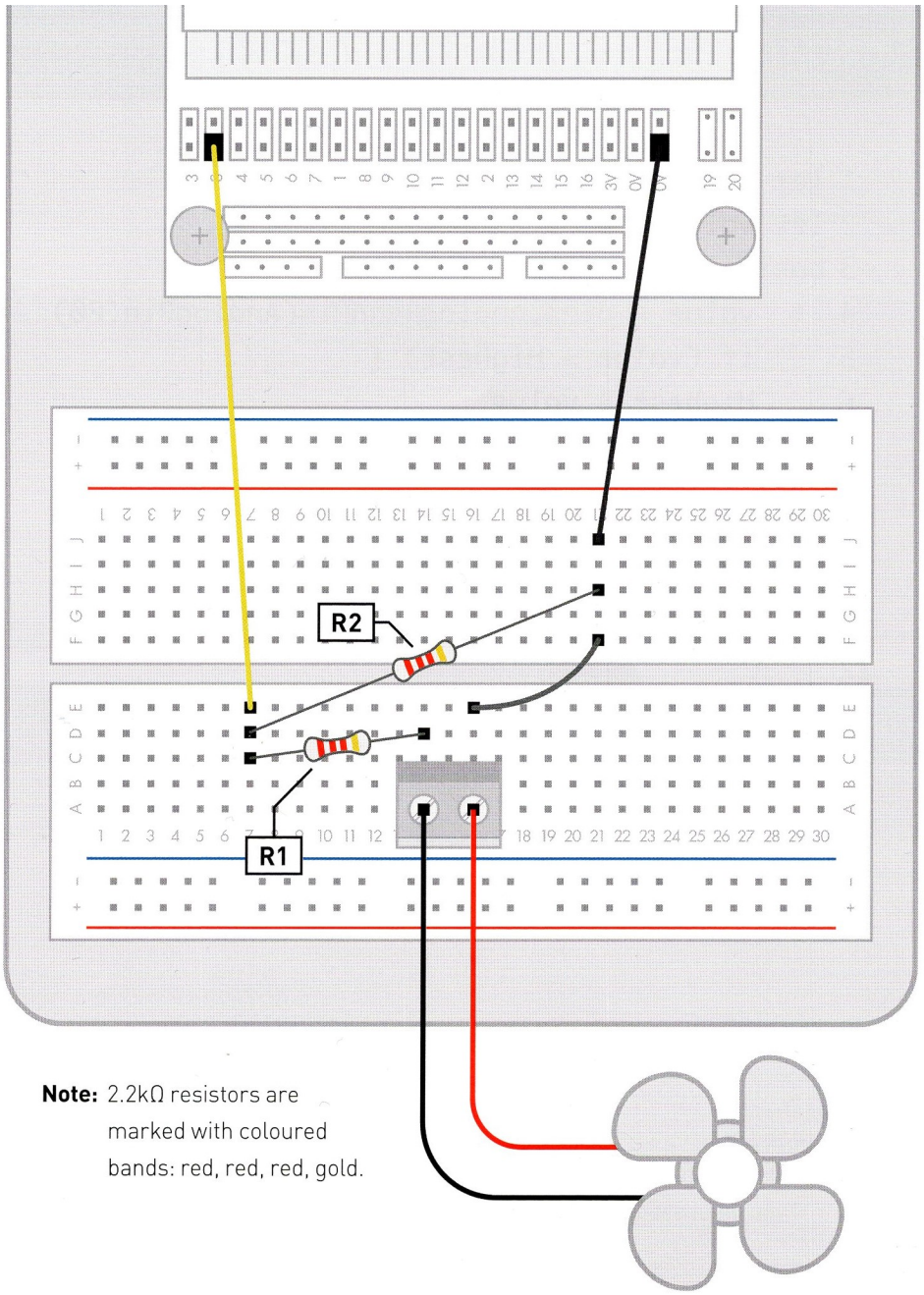
Commands are processed by transmitting in order the hashed command values, receiving them, and then comparing them against the commands hashed locally to see if the command present. If the pin is even one character different, it would be a completely different hash string creating privacy on tier with that created by AES.

A drawback of SHA256 is created by its lightweight nature, its easy to crack. But in the case of this implementation cracking the hash the first time around we simply yield a result that one might view as simple gibberish, as it utilizes the DPK which was also hashed in SHA256. One might also draw to attention that cracking this particular implementation may be more difficult than the average SHA256 string, due to the fact the DPK uses a user inputted pin, and pseudo-randomly generated salt, which is combined and hashed. The DPK is then combined with the command pins and hashed, meaning that would be crackers would have to crack a pseudo randomly generated string 256 bits long.

It is worth noting that this method is only viable due to the fact that AES doesn’t offer any security in the context of radio communication. A would be attacker still can perform all the attacks using the SHA256 method that they could perform in the AES method, but once again we iterate that the intention of the method in this context is not security, radio communication removes the possibility of sophisticated security methods and therefore leaves us with the goal of maximizing privacy instead.

This method is worth considering, it is lighter weight than AES and therefore would perform better on the Micro-bit. Furthermore we would be transmitting 256 bits of data rather than 128 bits of data, which would further obfuscation and privacy.

Circuit



Note: 2.2kΩ resistors are marked with coloured bands: red, red, red, gold.

References

Lancaster University (2016) *micro:bit Runtime.* Available from: https://lancaster-university.github.io/microbit-docs/ [Accessed 17 December, 2019]