Task 3 Specification: A simple device authentication system

Section 1: Overview of Assessment

This work is worth **40%** of the overall mark for the task 3, which make up 75% of the modules overall mark.

The assignment is aimed at introducing an IoT device authentication system with the *microbits Kitronik*. or Raspberry 4, or other devices like your mobile phone, laptop, etc.

The assignment is described in more detail in Section 2.

This is a group assignment:

(1) The group size is 2 or 3, but you need submit individually.
(2) Working on this assignment will help you to understand the IoT security, IoT systems, and programming with c/c++. If you have questions about this assignment, please post them to the discussion board hosted on Blackboard or email me.

This assignment must be submitted via, with the completion of a task 3 submission (source code and two documents), found on Blackboard, by 2pm on the 16rd Jan, 2020.

Section 2: Task Specification

This assignment aims to implement a simple authentication system that provides pin protected device access and encrypted data communication between two *microbits* or a *microbit* and another device, e.g. a laptop or phone, using either the simplified radio communication (uBit.radio), or the more advanced Bluetooth LE.

Your solution should be implemented using c or c++. Your solution must be committed in UWE's Gitlab. It will naturally be time stamped and you must be careful to not make commits after the submission deadline.

For the BLE solution you might want to look at using mBED rather that the microbit DAL runtime provided in the example.

Task 3 details:

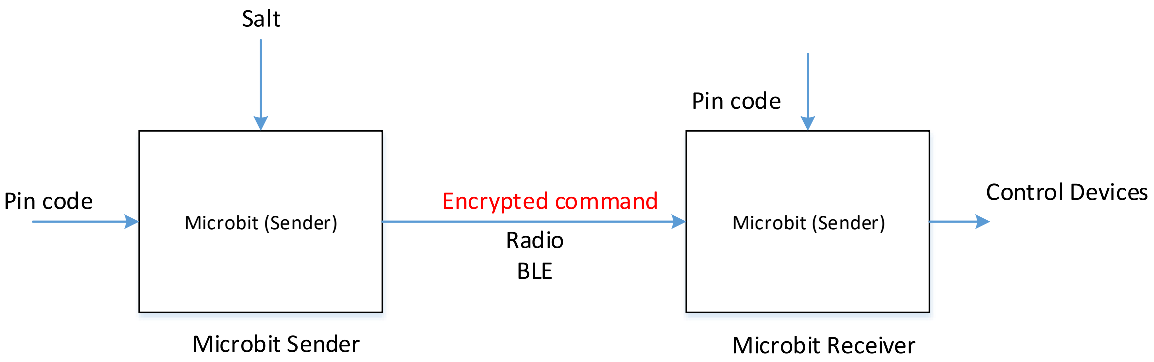
1. Define four or more commands for Kitronic kits/raspberry pi:

```
char *command[]={“motor”, “fan”,  “rgb_led_on”, “led_on”}
```

Define a pin code for each command. You can input the pin code with the microbit button(s), and your authentication system should be able to send encrypted command to another microbit.

2. Another microbit should be able to receive the encrypted commands, and using proper decryption method to decrypt the received information and extract the commands, then can run the command.

Possible ideas:



Sender:

- Generate a 128-bit **salt**
- Generate a data protection key, **dpk**=sha256(**pin**+**salt**)
- Use AES-ECB to encrypted the command **ciper**=aes_enc(**command**, **dpk**)
- Send the **cipher+salt** to another receiver microbit via radio or BLE

Receiver:

- Receive **cipher**, and input a **pin**,
- Generate **dpk**=sha256(**pin+salt**)
- Decrypt the cipher, **command**=aes_dec(**cipher**, **dpk**)
- Run the **command**

The communication example via radio can be found from https://lancaster-university.github.io/microbit-docs/ubit/radio/#ubitradio.

Your solution's source code should follow a coding convention, it should be well commented, and include a README.md on how to build it and what and how to use your solution.

Submission should be individually and the README.md must document who contributed to the submission.
More about the communication: here are some possible ideas:

- Using the simplified radio comms API, which is part of the DAL, so that it can work wireless. This should be fairly straightforward project and will be marked according.
- Using the simplified radio communicaiton API, which is part of the DAL, develop a protocol to communicate commands to work with the BBC microbit inventor kit.
- Node.js library to develop BLE central applications is: https://github.com/noble/noble. Note, your laptop or desktop device must support BLE (Bluetooth 4) for this to work.

Section 3: Deliverables

- The ***system solution***, e.g. c/c++ implementation, should be committed in your own directory in UWE's Gitlab, including a README.md documenting what the program does, how to build it, and any additional documents).
- The source code must consist of more than a single source file, including at least one header file, and two .cpp files.
- Your submission should include, in the README.md, a description of your protocol, including a **state diagram** of its working. If implement an alternative protocol that uses encryption, then your submission should include details of this and how it works.
- Submit a signed PDF ***system specification*** (see attached template) to Blackboard, which should include a link to your Git project for this module.
- An ‘Individual reflective statement’ (see attached template)

Section 4: Marking Criteria

Each challenge will be marked with the following criteria

	0-29	30-39	40-49	50-59	60-69	70-100	feedback
Functionality	Application delivers less than 30% of the required functionality	Application delivers 30-50% of required features as specified in the requirements	Application delivers 51 to 60% of required features as specified in the requirements	Application delivers 61 to 70% of required features as specified in the requirements	Application delivers over to 80% of required features as specified in the requirements	Application delivers beyond what was specified by the required features as specified in the requirements	
Performance analysis	Little or no analysis is provided .	Analysis does little to aid understanding of performance behaviour of both the provided serial program and implemented parallel variant.	Analysis presentation is not and precise. References to methodology used and background context for benchmarking is not detailed or even included.	Analysis presentation is clear and precise. References to methodology used and background context for benchmarking and so on but less than 70% complete	Analysis presentation is clear and precise. References to methodology used and background context for benchmarking and so on but less than 71% - 85% complete	Analysis presentation is clear and precise. References to methodology used and background context for benchmarking and so on and is more than 85% complete	
Implementation	Application uses example code with minimal changes OR no attempt has been made to use programming standards	Programming standards, application structures, and user interface design standards are not applied consistently	Application inconsistently applies programming standards and user interface design standards	Application consistently applies programming standards and user interface design standards	Application structure is clean and matches standards	Expertly uses the programming techniques and programming and interface design standard that were taught in the module.	
Internal Documentation **(Comments etc)**	No internal documentation	Little OR inconsistent internal documentation	Internal documentation does little to aid understanding of the code	Internal documentation is helpful but inconsistent with the standards taught AND insufficient	Internal documentation is helpful but inconsistent with the standards taught OR insufficient	Internal documentation is helpful and mostly consistent with the standards taught	