2 - Authentication

1. Learning Outcomes

On completion of this lab you will have:

- Implemented two different types of API authentication to protect access to a service backend
- Used database encryption features to implement password hashing generation and verification

2. Organisation

Please complete the exercises individually.

3. Grading

This worksheet is worth up to 10% of your overall module grade.

Note: You must attend and sign in at 6 labs in order to obtain full credit for your submitted worksheets. You may work on this worksheet during labs 3 and 4 with instructor assistance.

4. Submission

The deadline for submission is Sunday Mar 03, 2019 @23:59 through Github.

The work and submission workflow is as follows:

5. Before you start working on your worksheet you must **fork** a copy of the official class repo from here
    a. https://github.com/bjg/2019-tudublin-cmpu4023.git
6. Then **clone** the forked repo to your development machine
7. The make a new branch in your cloned, local repo named as follows:
    a. ***<student-id>***-wks-2

    where ***<student-id>*** is something like C12345678
8. When you are finished developing your worksheet solution then you must push your local repo to the remote origin for that branch

9. Finally, when you are submitting your solution for grading, you will generate a pull request (PR) requesting that your branch is merged with the remote origin master branch of the official class repo above
10. If you are not sure about any of the described steps here, then take a look at this worked demonstration:
    a. https://www.youtube.com/watch?v=FQsBmnZvBdc

11. Demonstration

You will demonstrate your solution to the lab instructor during the lab 5 session.

12. Requirements

For this lab you will need to

- Review the related module lecture material on Webcourses (lectures 14-16)

13. Resources

You are free to research whatever you need to solve the problems in this lab. Some recommended resources include:

- https://jwt.io/
- https://github.com/dwyl/learn-json-web-tokens
- https://github.com/joaquimserafim/json-web-token
- https://github.com/auth0/node-jsonwebtoken
- https://www.postgresql.org/docs/current/static/pgcrypto.html
- https://www.wolfe.id.au/2012/10/20/what-is-hmac-authentication-and-why-is-it-useful/

14. Problem Sets

The following platform-independent tasks can be solved on Windows, Mac local Linux or Cloud Linux as you prefer

Start with a blank NodeJS (*) project and PostgreSQL (*) database.

(*) Choose another API framework and database as you wish (with the usual caveats)

1	Implement a users table having a username and hashed password fields. Use the postgresql `crypt()` and `gen_salt()` functions to implement the password hashing Implement a protected resource table (e.g. a “products” table) to which you can use to demonstrate your authentication features	10 Marks
2	Implement a JWT-secured version of the API based on the users table from the previous step. Your solution will implement the following API extensions ● A (pre-authentication) login API call which accepts a username and password and returns (if successful) a JWT with a set of claims. The claims should include, minimally, the user id and an expiry timestamp; the token should be set to expire no later than 24 hours ● A mechanism to verify client tokens as bearer tokens in a HTTP Authorization header field ● Authentication should be applied, minimally, to any API calls which update any tables; Token validation should be performed on all API calls ● Assume the client has a priori knowledge of the user password ● Use asynchronous crypto in your solution Demonstrate your JWT authentication on a protected resource If authenticated or validated, the API return code should be in the 2xx range, otherwise 401.	40 Marks
3	Extend the users table or add another linked “apikeys” table to include an access key (160 bits) and secret key (320 bits)	10 Marks
4	Implement a Hash-based message authentication (HMAC) scheme to secure the API. In your solution you should include the following API message contents as part of the hashed/signed component: ● Message body (if any) ● Access key (prepended or appended as you choose) ● Query parameters (if any) Demonstrate your HMAC authentication on a protected resource If authenticated, the API return code should be in the 2xx range, otherwise 401.	40 Marks

	Note that to test hash-based authentication, you will need to create a simple client capable of generating valid signed requests	