2 - Authorisation

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 10 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 Feb 25, 2018 @23:59 through Webcourses.

5. Demonstration

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

6. Requirements

For this lab you will need to

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

7. 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/

8. 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	40 Marks

	update any tables; Token validation should be performed on all API calls ● Assume the client has a priori knowledge of the user password Demonstrate your JWT authentication on a protected resource If authenticated or validated, the API return code should be in the 2xx range, otherwise 401.	
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. **Note** that to test hash-based authentication, you will need to create a simple client capable of generating valid signed requests	40 Marks