**Faculty: Computer System Engineering**

**NAME: Thant Zin Soe**

**MODULE CODE: CET-324**

**MODULE TITLE: Advanced Cyber Security**

**MODULE LEADER: Matthew Banton**

**SUBMISSION DATE AND TIME: 5/April/2025 AND 20:33 GMT**

**ASSIGNMENT: Authentication Tokens**

**ASSESSMENT: 2 of 2**

**Academic Misconduct is an offence under university regulations, and this involves:**

* **Plagiarism** – where you use information from another information source (including your previously submitted work) and pass it off as your own. This can be through direct copying, poor paraphrasing and/or absence of citations.
* **Collusion** – where you work too closely, intentionally, or unintentionally, with others to produce work that is similar in nature. This can be through loaning of materials, drafts or through unauthorised use of a fellow student’s work.
* **Asking another person to write your assignment** – where you ask another individual or company to complete your work for you, be that paid or unpaid, and submit it as if it were your own.
* **Unauthorised use of artificial intelligence** – where you use artificial intelligence tools to generate your assignment instead of completing it yourself and/or where you have not been given permission to use artificial intelligence tools by your module leader. Please complete the following declaration around you use of artificial intelligence tools in your assignment.

**STATEMENT ON USE OF ARTIFICIAL INTELLIGENCE TOOLS:**

* I have used artificial intelligence tools to generate an idea for my assignment: **YES**
* I have used artificial intelligence tools to write my assignment for me: **NO**
* I have used artificial intelligence tools to brainstorm ideas for my assignment: **YES**
* I have used artificial intelligence tools to correct my original assignment: **NO**

**DECLARATION**

* I understand that by submitting this piece of work I am declaring it to be my own work and in compliance with the university regulations on Academic Integrity.
* I confirm that I have done this work myself without external support or inappropriate use of resources.
* I understand that I am only permitted to use artificial intelligence tools in line with guidance provided by my Module Leader, and I have not used artificial intelligence tools outside this remit.
* I confirm that this piece of work has not been submitted for any other assignment at this or another institution prior to this point in time.
* I can confirm that all sources of information, including quotations, have been acknowledged by citing the source in the text, along with producing a full list of the sources used at the end of the assignment.
* I understand that academic misconduct is an offence and can result in formal disciplinary proceedings.
* I understand that by submitting this assignment, I declare myself fit to be able to complete the assignment and I accept the outcome of the assessment as valid and appropriate.

**Table of Contents**

[1. Introduction 2](#_Toc194094379)

[2. Rationale for Design Decisions 2](#_Toc194094380)

[2.1 Programming Language: Python 2](#_Toc194094381)

[2.2 Web Framework: Flask 2](#_Toc194094382)

[2.3 Token Format: JSON Web Token (JWT) 3](#_Toc194094383)

[2.4 Token Payload Fields 3](#_Toc194094384)

[2.5 Token Security: HS256 Digital Signature 3](#_Toc194094385)

[2.6 Hashing Method: bcrypt 4](#_Toc194094386)

[2.7 Data Storage: Firebase Firestore 4](#_Toc194094387)

[2.8 Access Control: Role-Based Token Claims 5](#_Toc194094388)

[3. Code Snippets & Explanation (Enhanced) 5](#_Toc194094389)

[3.1 Password Hashing with bcrypt 5](#_Toc194094390)

[3.2 Login Verification using bcrypt 6](#_Toc194094391)

[3.3 JWT Token Generation with HS256 7](#_Toc194094392)

[3.4 Token Validation and Signature Check 8](#_Toc194094393)

[3.5 Firebase Integration for User Lookup 8](#_Toc194094394)

[4. Conclusion 9](#_Toc194094395)

[References 9](#_Toc194094396)

Table of Figures

[Figure 1Firebase database users' data collection structure 6](#_Toc194096755)

[Figure 2:Register account (plain text password) 7](#_Toc194096756)

[Figure 3:Hashed password is stored in database 8](#_Toc194096757)

[Figure 4Log in with correct password ( valid token ) 8](#_Toc194096758)

[Figure 5: Token generated after login. 9](#_Toc194096759)

[Figure 6:Decoded token validation result with expiry and role. 10](#_Toc194096760)

# 1. Introduction

This report outlines the design and implementation of a secure token-based authentication system for a client-server application. It includes user registration, login, and token issuance for access control. The system uses modern cryptographic best practices like password hashing and token signing to ensure forgery and tampering resistance. Built using Python and Flask, and Firebase Firestore as cloud storage, it achieves a good balance between security, trust, and usability while adhering to cybersecurity best practices.

# 2. Rationale for Design Decisions

## 2.1 Programming Language: Python

Python was selected due to its simplicity, readability, and robust support in web development and cybersecurity. It is rich in libraries to make it simple to include routing, encryption, hashing, and database functionality. Python's popularity also makes maintenance and collaboration easier(Downey, 2015).

## 2.2 Web Framework: Flask

Flask was chosen for its lightweight and flexible nature, offering full control over system architecture. It allows easy configuration for both local and cloud setups and includes only essential components like routing and templates—making it ideal for building a custom authentication system.(Grinberg, 2018).

## 2.3 Token Format: JSON Web Token (JWT)

JWT was chosen because of its compact size, simplicity, and the fact that it supports stateless, secure authentication. Being widely used in web systems, it contains a header, payload, and signature—validation can be achieved without database calls. Its capability to handle expiry time, issue time, and custom claims was appropriate for the task requirements (Tragura, 2024).

## 2.4 Token Payload Fields

The JWT token generated by the system includes several fields to ensure both functionality and security:

* email: identifies the user who owns the token
* access: defines their role (e.g., user, admin)
* system: names the issuing system
* exp: expiry timestamp (15 minutes from issuance)

These fields allow for strong access control and traceability without storing session data on the server. Including exp ensures the token becomes invalid after a set period, minimizing the risk of token theft being exploited.

## 2.5 Token Security: HS256 Digital Signature

To secure against tampering, each token is signed and checked using the HS256 algorithm (HMAC with SHA-256). It uses a secret key to create a digital signature, proving it as authentic. When the token is modified, the verification will fail. HS256 is popular as it is both secure and efficient, providing light but strong protection (Michał, 2019 ).

## 2.6 Hashing Method: bcrypt

Passwords are hashed using bcrypt, a slow, salted hash function that is resistant to brute-force and rainbow table attacks. In contrast to fast hashes like SHA-1, bcrypt ensures that each hash is unique. The system uses bcrypt.hashpw() upon registration and bcrypt.checkpw() upon login to validate passwords without saving them in plain text (Seth, 2019).

## 2.7 Data Storage: Firebase Firestore

The system takes advantage of Firebase Firestore, a secure and adaptive NoSQL cloud database, to hold the user credentials and roles. The system is compatible with Python through the firebase-admin SDK and provides real-time updates with access control. Firestore, unlike local storage, is always available and great for scaling web and mobile applications (Yahiaoui, 2017).

A screenshot of a computer

AI-generated content may be incorrect.

Figure Firebase database users' data collection structure

## 2.8 Access Control: Role-Based Token Claims

The access level of every user, i.e., user or admin, is kept in Firestore and included in the JWT as an access claim. This facilitates Role-Based Access Control (RBAC). This allows future parts of the system to authorize actions based on roles without needing an extra session system. This allows for both flexibility and trust (Sneha, 2024).

# 3. Code Snippets & Explanation (Enhanced)

## 3.1 Password Hashing with bcrypt

|  |
| --- |
| hashed = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')  users\_ref.document(email).set({  'password': hashed,  'access': 'user'  }) |

A white rectangular object with blue lines

AI-generated content may be incorrect.

Figure :Register account (plain text password)

A screenshot of a computer

AI-generated content may be incorrect.

Figure :Hashed password is stored in database

**Explanation**:  
Passwords are never stored in plaintext by the user. Rather, they're hashed using bcrypt, which defends against brute-force and rainbow table attacks using a slow, salted hash. Even when Firebase is compromised, the original passwords are still safe following the principle of reducing data exposure.

## 3.2 Login Verification using bcrypt

|  |
| --- |
| stored\_hash = user\_doc.to\_dict()['password'].encode('utf-8')  return bcrypt.checkpw(password.encode('utf-8'), stored\_hash) |

A screenshot of a phone

AI-generated content may be incorrect.

Figure Log in with correct password ( valid token )

**Explanation**:  
When a user logs in, their submitted password is compared against the stored hashed version using bcrypt.checkpw(). This function handles the internal salt automatically and ensures that the check is secure. This implementation avoids storing passwords or even reversible hashes, aligning with best practices in secure authentication systems.

## 3.3 JWT Token Generation with HS256

|  |
| --- |
| payload = {  'email': email,  'access': user\_data['access'],  'system': 'FlaskAuthApp',  'iat': datetime.datetime.utcnow(),  'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=15)  }  token = jwt.encode(payload, JWT\_SECRET, algorithm=JWT\_ALGORITHM) |

A screen shot of a computer

AI-generated content may be incorrect.

Figure : Token generated after login.

**Explanation**:  
On successful login, the system generates a secure JWT containing the user’s email, role, system name, and a 15-minute expiry. It is signed with HS256, preventing tampering or forgery without the server’s secret key—ensuring token integrity and meeting assignment requirements.

## 3.4 Token Validation and Signature Check

|  |
| --- |
| decoded = jwt.decode(token, JWT\_SECRET, algorithms=[JWT\_ALGORITHM]) |

A screenshot of a computer

AI-generated content may be incorrect.

Figure :Decoded token validation result with expiry and role.

**Explanation**:  
When accessing protected resources, the system verifies the token’s signature using the original key. These blocks replay attacks and ensure only valid, untampered tokens are accepted. Expired or altered tokens fail verification, enforcing trust and session security.

## 3.5 Firebase Integration for User Lookup

|  |
| --- |
| user\_doc = users\_ref.document(email).get()  if user\_doc.exists:  return user\_doc.to\_dict() |

**Explanation**:  
User data is securely retrieved from Firebase Firestore, a scalable NoSQL cloud database. With server-side access and security rules, roles and hashed credentials are protected, supporting scalability and secure authentication without managing local storage.

# 4. Conclusion

It employs secure and versatile token-based access control with Python, Flask, JWT, and bcrypt to maintain confidentiality and integrity of information. It controls role-based access and verifies tokens without storing sessions, making it suitable for usability and scalability. Firebase provides cloud readiness, ensuring a balance between cybersecurity regulations and usability.

# References

Downey, A. (2015). *Think Python: How to Think Like a Computer Scientist.* (Vol. 2nd ed). Needham, MA: Green Tea Press. Retrieved from https://books.ms/main/CEEF7F4C0E59C6EBD657EA9F6328CB7A

Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python.* Sebastopol, CA: O'Reilly Media. Retrieved from https://books.ms/main/30322D78018BBF481481B2D656F5C2E0

Michał, S. (2019 ). *JWT (JSON Web Token) (in)security.* securitum. Retrieved from https://research.securitum.com/jwt-json-web-token-security/

Seth, J. N. (2019). *Practical Cryptography in Python* (Vol. 1 st edition). Austin, TX, USA: APRESS. Retrieved from https://link.springer.com/content/pdf/bfm:978-1-4842-4900-0/1

Sneha. (2024, Dec 25). *Implementing Role-Based Access Control (RBAC) with JWT.* Retrieved from Byte of Knowledge: https://medium.com/byte-of-knowledge/implementing-role-based-access-control-rbac-with-jwt-672805628f83#:~:text=Role%2DBased%20Access%20Control%20(RBAC)%20is%20a%20security%20paradigm,and%20authorization%20in%20web%20applications.

Tragura, S. J. (2024). *Mastering JSON Web Tokens: Secure APIs and Web Apps* (Vol. 1st ed). Packt Publishing Pvt Ltd,. Retrieved from https://books.ms/main/B6CEC60A3A221B8E8BB8366867B4B8D6

Yahiaoui, H. (2017). *Firebase Cookbook.* Packt Publishing, . Retrieved from https://books.ms/main/F1513FC4106056A140460F47775B66BD