# **CSCI391 - ST: Microservices Lab 6 - JWT**

## ****Objective****

In this lab, you'll explore how **JSON Web Tokens (JWTs)** work by:

1. Completing an interactive tutorial.
2. Implementing core JWT-related cryptographic methods.
3. Solving the 🔥 **Bonus Challenge: The Token of Doom** 🔥.

## ****🧪 Part 1: Complete the Online Lab****

Go to:  
🔗 <https://developer.auth0.com/resources/labs/tools/jwt-basics#introduction>

Complete all interactive sections:

* JWT structure (Header, Payload, Signature)
* Base64Url encoding
* HMAC-SHA256 signatures
* Signature validation

Take notes—you'll use these concepts shortly.

## ****💻 Part 2: Implement Cryptography****

Instead of downloading code, you'll work directly with the following files.

### 📄 pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>

<artifactId>JWT</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>23</maven.compiler.source>

<maven.compiler.target>23</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.bouncycastle</groupId>

<artifactId>bcprov-jdk15on</artifactId>

<version>1.70</version>

</dependency>

</dependencies>

</project>

📄 ICryptography.java

package org.example;

public interface ICryptography {

String hmacSHA256(String message, String secret);

String encode(String rawString);

String decode(String encodedString);

}

📄 Cryptography.java

package org.example;

import javax.crypto.Mac;

import javax.crypto.spec.SecretKeySpec;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.util.Base64;

public class Cryptography implements ICryptography {

// STUDENT TODO: Implement encode

public String encode(String rawString) {

// Remove newlines and Base64 URL-safe encode

throw new UnsupportedOperationException("encode() not implemented");

}

// STUDENT TODO: Implement decode

public String decode(String encodedString) {

// Base64 decode and return plain string

throw new UnsupportedOperationException("decode() not implemented");

}

private String base64UrlEncode(byte[] input) {

return Base64.getUrlEncoder().withoutPadding().encodeToString(input);

}

@Override

public String hmacSHA256(String message, String secret) {

try {

Mac sha256HMAC = Mac.getInstance("HmacSHA256");

SecretKeySpec secretKey = new SecretKeySpec(secret.getBytes(), "HmacSHA256");

sha256HMAC.init(secretKey);

byte[] macData = sha256HMAC.doFinal(message.getBytes());

return base64UrlEncode(macData);

} catch (NoSuchAlgorithmException | InvalidKeyException e) {

throw new RuntimeException("Error calculating HMAC-SHA256", e);

}

}

}

📄 CryptographyTests.java

import org.example.Cryptography;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CryptographyTests {

private Cryptography cryptography;

@Before

public void setUp() {

this.cryptography = new Cryptography();

}

@Test

public void testDecode() {

String encodedString = "cCI6IkpXVCJ9";

String decodedString = cryptography.decode(encodedString);

assertEquals("p\":\"JWT\"}", decodedString);

}

@Test

public void testEncode(){

String rawString = "{\"alg\":\"HS256\",\"typ\":\"JWT\"}";

String encodedString = cryptography.encode(rawString);

assertEquals("eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9", encodedString);

rawString = "{\"name\":\"DevDay23\",\"iat\":1684245600}";

encodedString = cryptography.encode(rawString);

assertEquals("eyJuYW1lIjoiRGV2RGF5MjMiLCJpYXQiOjE2ODQyNDU2MDB9", encodedString);

}

@Test

public void testHmacSHA256() {

String headerEncoded = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9";

String payloadEncoded = "eyJuYW1lIjoiRGV2RGF5MjMiLCJpYXQiOjE2ODQyNDU2MDB9";

String secret = "useauth0byoktatobuildyourcustomidentitypipeline";

String data = headerEncoded + "." + payloadEncoded;

String signature = cryptography.hmacSHA256(data, secret);

String jwtToken = data + "." + signature;

System.out.println(jwtToken);

assertEquals("PtEH73\_fTnj4vbd6js8G66H3Me69RqSE-0w3bevBfNE", signature);

}

}

**🔥 Bonus Challenge: The Token of Doom 🔥**

*"One token to rule them all, one token to find them,*  
*One token to bring them all, and in the payload bind them…*  
*In the land of JWT, where secrets lie."*

You’ve been entrusted with this JWT:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJoaW50IjoiaHR0cHM6Ly93d3cueW91dHViZS5jb20vd2F0Y2g\_dj0tdEpZTi1lRzF6ayJ9.JCDI5aHlUEs6avrcpTOJ43YPQwcbPaFbJPpXBDaKCDc

**🔥 Instructions**

1. **Decode the payload** using your decode() implementation.
2. Discover the **hint** in the decoded payload.
3. Guess potential **secrets** (start with the obvious hint).
4. Use your encode() and hmacSHA256() methods to recreate the signature.
5. Once the signature matches, you’ve found the correct **secret**.

Please include your implementation of Cryptography.java decode() and encode() methods, as well as your guess for the secret for your submission on Canvas😊