REST API Security
API Keys and Java Web Tokens

Guadalupe Ortiz Bellot

Computer Science and Engineering Department

UCA
Universidad de Cádiz

Contents

1. API Keys
2. JWT
3. JWE

Contents

1. **API Keys**
2. JWT
3. JWE

1. API KEYS Introduction (i)

- When we need to authenticate from another application without accessing the data hosted in the application.
- How API Keys work:
- An API key assigns a unique 128-bit value to a user of the RESTful service.
- This API key is associated to a user and maintained in a datastore.
- The RESTful service then references this datastore together with the service id (optional) and API-KEY with each incoming request.
- Once the incoming requests are validated, access is either granted or denied to the specified end-point.

1. API KEYS Introduction (ii)

Advantages

- No need to transmit user password during authorization between client and server
- API-KEYs are faster than Digest Authentication as only one call is needed as opposed to 2 calls for every request
- API-KEYs works well with TLS/SSL and should be included in the header for added protection and to prevent the values from being captured in the logs.

1. API KEYS
Where to place the api key

- API-KEY / Service ID Required in Header
- API-KEY and Service ID should be included in the HTTP Headers (@HeaderParam) instead of HTTP Parameters via (@QueryParam).
- This is especially important when using TLS/SSL as it will guarantee that request data is encrypted end to end and prevent man in the middle attacks.

1. API KEYS Implementation (i)

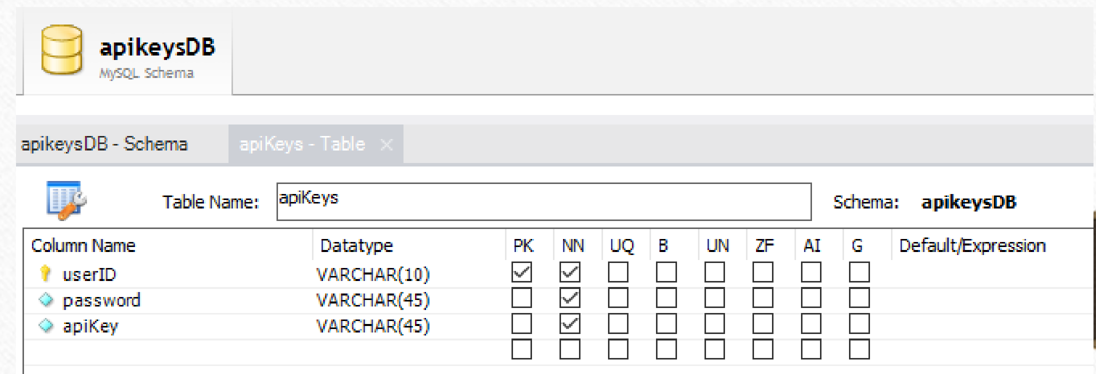
- Create a User class with 3 private attributes and their correspnding set/get methods: user, password, apikey

- Optionally create database and table for users
- Alternatively create a static structure in the Hello class

1. API KEYS Implementation (ii)

With database:

- Include mysql driver jar in the project lib folder (already included)
- Create User class (with attributes userD, password and apikey and the corresponding get and set)
- Create DBAccess class with methods to create/update/obtain/delete users



With static structure:

```
private static Map<String, User> myUserDB = new HashMap<>();
```

1. API KEYS Implementation (iii)

New user obtain an apikey:

```
@POST
@Path("/apikey")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.TEXT_PLAIN)
```

```
public String getApiKey(User myUser)
{
    UUID apikey = UUID.randomUUID();
    User newUser = new User();
    newUser.setUser(myUser.getUser());
    newUser.setPassword(myUser.getPassword());
    newUser.setApikey(apikey.toString());
    myUserDB.put(myUser.getUser(),newUser);
    return apikey.toString();
}
```

1. API KEYS Implementation (iv)

- Access with the apikey
- You can add password verification

```
@POST
@Path("/testApikey1")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.TEXT_PLAIN)
```

```
public String testing (User myUser, @HeaderParam("apikey") String apikey)
{
  if (myUserDB.containsKey(myUser.getUser())) {
  if ((myUserDB.get(myUser.getUser()).getApikey()).equals(apikey)) {
  return "GRANTED";}
    }
  return "Denied";
}
```

1. API KEYS Testing (i)

- Obtain Apikey (i)

POST http://localhost:8080/HeloWorld/demo/hello/apikey Send

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings | Cookies

Type

Digest Auth

The authorization header will be automatically generated when you send the request.
Learn more about authorization ↗

By default, Postman will extract values from the received response, add it to the request,

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.
Learn more about variables ↗

Username: restUser2

Password: ••••••••••••••••

Show Password

POST http://localhost:8080/HeloWorld/demo/hello/apikey Send

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings | Cookies

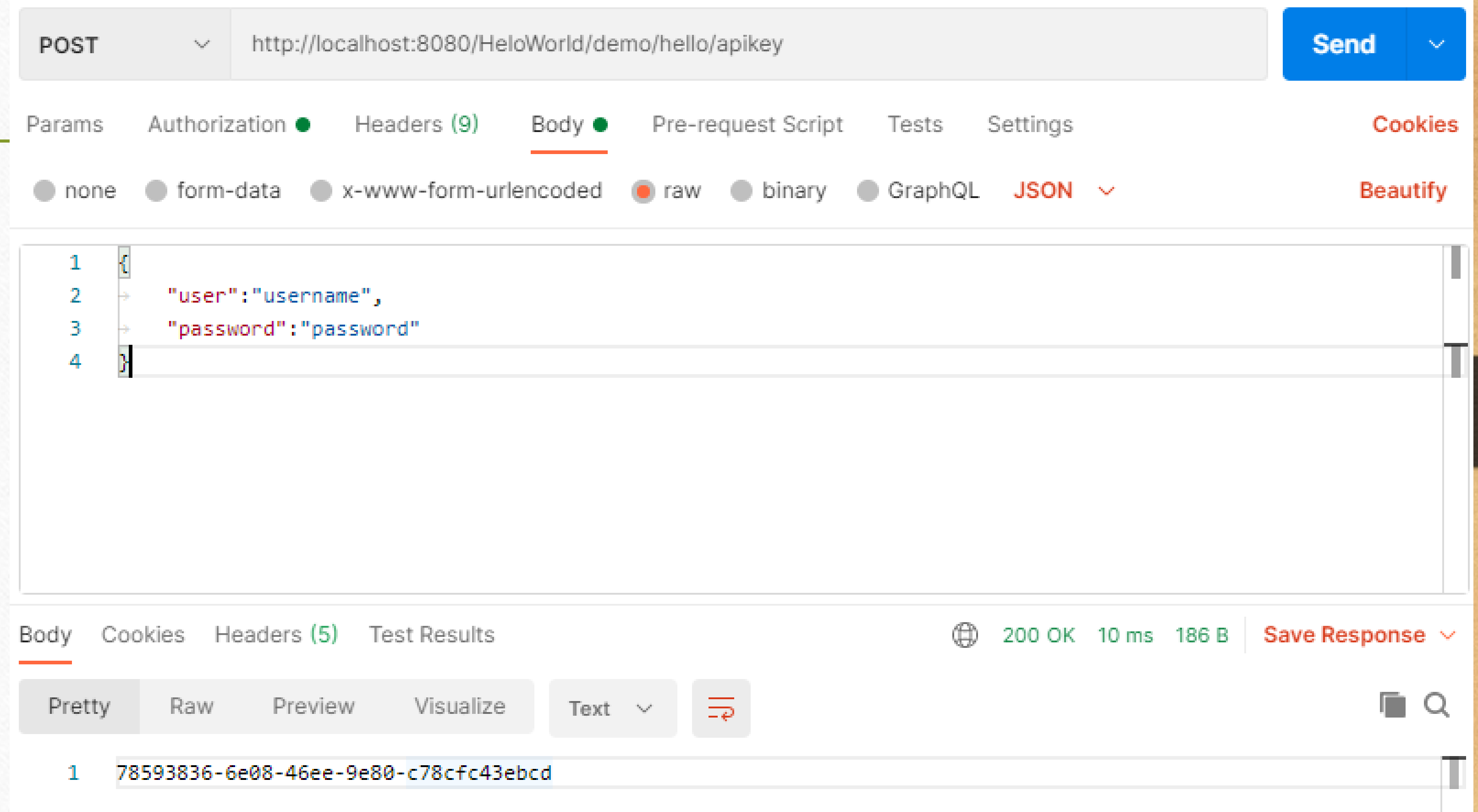
Headers 8 hidden

	KEY	VALUE	DESCRIPT
☑	Content-Type	application/json	
	Key	Value	Description

1. API KEYS Testing (ii)

- Obtain Apikey (ii)

- Copy the api key obtained.



1. API KEYS Testing (iii)

- Validate apikey (i)
- Paste the copied apikey as a new header parameter.

POST http://localhost:8080/HeloWorld/demo/hello/testApikey1 Send

Params Auth Headers (10) Body Pre-req. Tests Settings Cookies

Type

Digest Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

By default, Postman will extract values from the received response, add it to

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

Username restUser2

Password ••••••••••••••••

Show Password

POST http://localhost:8080/HeloWorld/demo/hello/testApikey1 Send

Params Auth Headers (10) Body Pre-req. Tests Settings Cookies

Headers 8 hidden

	KEY	VALUE	DESCRIP
☑	Content-Type	application/json	
☑	apikey	78593836-6e08-46ee-9e80-c78cfc4...	
	Key	Value	Description

Bulk Edit Presets

1. API KEYS Testing (iv)

- Validate apikey (ii)

POST http://localhost:8080/HeloWorld/demo/hello/testApikey1 Send

Params | Auth | Headers (10) | Body | Pre-req. | Tests | Settings | Cookies

raw | JSON | Beautify

```
1 {
2     "user":"username"
3
4 }
```

Body | Cookies | Headers (5) | Test Results | 200 OK | 9 ms | 156 B | Save Response

Pretty | Raw | Preview | Visualize | Text

```
1 GRANTED
```

Contents

1. API Keys
2. **JWT**
3. JWE
4. References

JSON WEB TOKENS (JWT) Introduction

- Authentication through a signed token which can be verified by application servers.
- JSON web tokens contain information that is unique for a user.
- To ensure that the token has not been altered the token contains a cryptographically encrypted digital signature.

JSON WEB TOKENS (JWT)
Structure

- JWT Tokens are encoded with base64 and signed with SHA-256.
- They are composed of a header, a payload and a signature.
  - Header: it contains the key id or jwt type and the hashing algorithm.
  - Payload: it contains the claims (registered (http://www.iana.org/assignments/jwt/jwt.xhtml),public and private).
  - Signature: it is composed of a base64 encoded header and payload making a hash using some type of Message Authentication Code (MAC).

JSON WEB TOKENS (JWT) Communication Flows

1. Initial request with username and password
2. Response with 200 status code and JSON Web Token
3. Following requests include the JWT in the header
4. Responses with 200 status code and response

JSON WEB TOKENS (JWT) Implementation (i)

- Include the libraries (already included)
- Create a new user type or you can reuse the one you already have
- Create a static JSON Web Key

```
static JsonWebKey myJwk = null;
```

- Create a method to *authenticateCredentials*
- Create a method to *testJWT*
- Pay attention to the new imports required from ***org.jose4j*** *and* ***com.fasterxml.jackson***

More info about public claims: https://www.iana.org/assignments/jwt/jwt.xhtml

JSON WEB TOKENS (JWT)
Authenticate credentials (i)

```
@Path("/authenticateJWT")
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response authenticateCredentials(@HeaderParam("username") String username,
        @HeaderParam("password") String password)
    throws JsonGenerationException, JsonMappingException, IOException {
```

```
User user = new User();
 user.setUser(username);
 user.setPassword(password);

RsaJsonWebKey jwk = null;
 try {
   jwk = RsaJwkGenerator.generateJwk(2048);
   jwk.setKeyId("1");
   myJwk=jwk;
 } catch (JoseException e) {   e.printStackTrace();  }
```

JSON WEB TOKENS (JWT)
Authenticate credentials (ii)

```
JwtClaims claims = new JwtClaims();
claims.setIssuer("uca");
claims.setExpirationTimeMinutesInTheFuture(10);
claims.setGeneratedJwtId();
claims.setIssuedAtToNow();
claims.setNotBeforeMinutesInThePast(2);
claims.setSubject(user.getUser());
claims.setStringListClaim("roles", "restUser2");
JsonWebSignature jws = new JsonWebSignature();
jws.setPayload(claims.toJson());
jws.setKeyIdHeaderValue(jwk.getKeyId());
jws.setKey(jwk.getPrivateKey());
jws.setAlgorithmHeaderValue(AlgorithmIdentifiers.RS
A_USING_SHA256);

String jwt = null;
try { jwt = jws.getCompactSerialization();
} catch (JoseException e) {System.out.println (e);}
user.setApikey(jwt); //SET TOKEN
return Response.status(200).entity(jwt).build();
```

JSON WEB TOKENS (JWT)
Test JWT (i)

```
@POST
@Path("/testJWT")
@Consumes(MediaType.TEXT_PLAIN)
@Produces(MediaType.TEXT_PLAIN)
public Response testJWT
(@HeaderParam("token") String token, String
myName)
        throws JsonGenerationException,
        JsonMappingException, IOException {
```

```
JsonWebKey jwk = myJwk;
// Validate Token's authenticity and check claims
JwtConsumer jwtConsumer = new JwtConsumerBuilder()
        .setRequireExpirationTime()
        .setAllowedClockSkewInSeconds(30)
        .setRequireSubject()
        .setExpectedIssuer("uca")
        .setVerificationKey(jwk.getKey())
        .build();
```

JSON WEB TOKENS (JWT)
Test JWT (ii)

```
try {
    //  Validate the JWT and process it to the Claims
    JwtClaims jwtClaims = jwtConsumer.processToClaims(token);
    System.out.println("JWT validation succeeded! " + jwtClaims);
  } catch (InvalidJwtException e) {
  return Response.status(Status.FORBIDDEN.getStatusCode()).entity("Forbidden").build();
  }
  String sayHello="Hello "+myName;
  return Response.status(200).entity(sayHello).build();


}
```

JSON WEB TOKENS (JWT)

Testing JWT (i)

- Get authentication credentials
- Copy the obtained token

GET http://localhost:8080/HeloWorld/demo/hello/authenticateJWT Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Headers 6 hidden

	KEY	VALUE	DESCRIP
☑	username	username	
☑	password	password	
	Key	Value	Description

Body Cookies Headers (5) Test Results — 200 OK 366 ms 714 B Save Response

Pretty Raw Preview Visualize JSON

1 eyJraWQiOiIxIiwiYWxnIjoiUlMyNTYifQ.
eyJpc3MiOiJ1Y2EiLCJleHAiOjE2MTQ4NTcxMzEsImp0aSI6IndjcHFTYUlYRVVyVHlLTWpMRzdiMVEiLCJpYXQiOjE2MTQ4NTY1MzE
sIm5iZiI6MTYxNDg1NjQxMSwic3ViIjoidXNlcm5hbWUiLCJyb2xlcyI6WyJyZXN0VXNlcjIiXX0.
NQ4viJNeRYLxGObpNKlk9W2RLJtA2aTY4qiEJBh-So6Z3fzgsEURErgLVeXnfQao8kEJQUGAVH9i4v2Mkz59RBQ8-s-_6uCSGxRGpl
m0hk5X1stp23B0L0jsk98tv4nTMa4_T6nKpLYu7H1PchO4RNc3FJcrM3LPRzsKBf73Upc3EBnnunTxj-Rm9RKQbS5--PHh6p8fDWo9_
WWq_ta3vGLJmrO9s2tICXbl6oayzQ43UN8nFLEpKGSxHBIlycTOSTp6zmuwZrrlWuYKsAW3Eg8NqPVTPfpDxXKglkleNUGz2pUziIkM
3dcG_cAgDccMLdnrSBi-kohbdVq-aYMmA

JSON WEB TOKENS (JWT)

Testing JWT (ii)

- Authenticate with JWT(i):
- Paste the obtained token in a new header param

POST http://localhost:8080/HeloWorld/demo/hello/testJWT Send

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings | Cookies

Type: Digest...

The authorization header will be automatically generated when you send the request.
Learn more about authorization ↗

By default, Postman will extract values from the received response, add it to the request,

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

Username: restUser2

Password: ••••••••••••••••

Show Password

POST http://localhost:8080/HeloWorld/demo/hello/testJWT Send

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings | Cookies

Headers 8 hidden

	KEY	VALUE	DESCRIPTIO
☑	token	eyJraWQiOiIxIiwiYWxnIjoiUlMyNTYifQ.eyJp...	
	Key	Value	Description

Bulk Edit | Presets

JSON WEB TOKENS (JWT)
Testing JWT (iii)

- Authenticate with JWT(ii):

POST http://localhost:8080/HeloWorld/demo/hello/testJWT Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL Text

```
1 Guadalupe
```

Body Cookies Headers (5) Test Results 200 OK 59 ms 165 B Save Response

Pretty Raw Preview Visualize Text

```
1 Hello Guadalupe
```

Contents

1. API Keys
2. JWT
3. **JWE**

JSON WEB ENCRYPTION (JWE) Introduction

- JSON Web Encryption:
- It is encrypted using cryptographic algorithms and serialized for tokenization in HTTP authorization headers.
- In order to ensure the message or token has not been altered in any way the token contains a digital signature (JWS) that is cryptographically encrypted using a strong algorithm such as HMAC SHA-256.

JSON WEB ENCRYPTION (JWE) Communication Flow

1. Initial request with username and password
2. Response with 200 status code and encrypted JSON Web Token
3. Following requests include the JWE in the header
4. Responses with 200 status code and response

JSON WEB ENCRYPTION (JWE) Implementation

- Create required static variables
- Create the method to obtain the token
- Create the method to be invoked with the token
- Pay attention to the new imports from *org.jose.jwe*

JSON WEB ENCRYPTION (JWE)
Create the encrypted key

```
static JsonWebKey jwKey = null;


static {
  // Setting up Direct Symmetric Encryption and Decryption
  String jwkJson = "{\"kty\":\"oct\",\"k\":\"9d6722d6-b45c-4dcb-bd73-2e057c44eb93-928390\"}";
  try {
    new JsonWebKey.Factory();
    jwKey = Factory.newJwk(jwkJson);
  } catch (JoseException e) {
    e.printStackTrace();
  }
}
```

JSON WEB ENCRYPTION (JWE) Authenticate credentials (i)

```
@Path("/authenticateJWE")
@POST
@Produces(MediaType.APPLICATION_JSON)
public Response authenticateCredentialsJWE(
    @HeaderParam("username") String username,
    @HeaderParam("password") String password)
    throws JsonGenerationException, JsonMappingException,
    IOException {

  User user = new User();
  user.setUser(username);
  user.setPassword(password);
```

```
JwtClaims claims = new JwtClaims();
claims.setIssuer("uca");
claims.setExpirationTimeMinutesInTheFuture(10);
claims.setGeneratedJwtId();
claims.setIssuedAtToNow();
claims.setNotBeforeMinutesInThePast(2);
claims.setSubject(user.getUser());
claims.setStringListClaim("roles", "admin");

JsonWebSignature jws = new JsonWebSignature();
```

JSON WEB ENCRYPTION (JWE) Authenticate credentials (ii)

```
jws.setPayload(claims.toJson());
  jws.setKeyIdHeaderValue(jwKey.getKeyId());
  jws.setKey(jwKey.getKey());
  jws.setAlgorithmHeaderValue(AlgorithmIdentifiers.HMAC_SHA256);

  String jwt = null;
  try {
    jwt = jws.getCompactSerialization();
  } catch (JoseException e) {    e.printStackTrace();  }

JsonWebEncryption jwe = new JsonWebEncryption();
```

JSON WEB ENCRYPTION (JWE) Authenticate credentials (iii)

```
jwe.setAlgorithmHeaderValue(
    KeyManagementAlgorithmIdentifiers.DIRECT);
jwe.setEncryptionMethodHeaderParameter(

ContentEncryptionAlgorithmIdentifiers.AES_128_CBC_HMAC_SHA_256);
jwe.setKey(jwKey.getKey());
jwe.setKeyIdHeaderValue(jwKey.getKeyId());
jwe.setContentTypeHeaderValue("JWT");
jwe.setPayload(jwt);
```

```
String jweSerialization = null;
try {
  jweSerialization = jwe.getCompactSerialization();
} catch (JoseException e) {   e.printStackTrace();  }

return Response.status(200).entity(jweSerialization).build();
}
```

JSON WEB ENCRYPTION (JWE)
Test JWE (i)

```
@POST
@Path("/testJWE")
@Consumes(MediaType.TEXT_PLAIN)
@Produces(MediaType.TEXT_PLAIN)
public Response testJWE (@HeaderParam("token") String token, String myName)
throws JsonGenerationException,
JsonMappingException, IOException {
```

```
JwtConsumer jwtConsumer = new JwtConsumerBuilder()
        .setRequireExpirationTime()
        .setAllowedClockSkewInSeconds(30)
        .setRequireSubject()
        .setExpectedIssuer("uca")
        .setDecryptionKey(jwKey.getKey())
        .setVerificationKey(jwKey.getKey()).build();
```

JSON WEB ENCRYPTION (JWE)
Test JWE (ii)

```
try {
    // Validate the JWT and process it to the Claims
    JwtClaims jwtClaims = jwtConsumer.processToClaims(token);
    System.out.println("JWE validation succeeded! " + jwtClaims);
  } catch (InvalidJwtException e) {
  System.out.println("JWE is Invalid: " + e);
    return Response.status(Status.FORBIDDEN.getStatusCode()).entity("Forbidden").build();
  }
  String sayHello="Hello "+myName;
  return Response.status(200).entity(sayHello).build();

}
```

JSON WEB ENCRYPTION (JWE) Testing JWE (i)

- Get authentication credentials (i)

POST http://localhost:8080/HeloWorld/demo/hello/authenticateJWE Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

Type

Digest Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

By default, Postman will extract values from the received response, add it to

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

Username restUser2

Password •••••••••••••••

Show Password

JSON WEB ENCRYPTION (JWE)

Testing JWE (ii)

- Get authentication credentials (ii)
- Copy the obtained token

POST http://localhost:8080/HeloWorld/demo/hello/authenticateJWE Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

Headers 7 hidden

	KEY	VALUE	DESCRIPTION
☑	username	username	
☑	password	password	
	Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK 29 ms 625 B Save Response

Pretty Raw Preview Visualize JSON

1 eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2Iiwia2lkIjpudWxsLCJjdHkiOiJKV1QifQ..nOKDPLPMPEiAbVZlNBEbmA.
-UrObMsPloTRwnxskCx_EeAMgw3ugXHBfFO4bTan5C88Xbxe7wxCSkeKy_VnZ5xWOV45XjVu8Pz0Y1AN3DGSKq6O8ttJkctCvSTIw
6eBbUKJYqlze4HIR8VWoSftEF7_dFTUL4Q6up9_Loje52aBiwtSiPiolABIftBDhd0UsuTGK8_wa0qXaMh0uVF0JGdzGt8PJ4kwSV
mr-cj4N-FP5WrA6MJ5hNpPRH6S8puBSh6wxunKp5e4xngjsRINoxqMZu4-V9-18yU-DDG100zaEGm9st4xvLgN1mWRHQNT0sciG03
r0Jh_9HTGGR7szKVKjv4a5Z-VCkk8X2YXoIJlFA.GqQc_dI-88bojlniSi9CQA

JSON WEB ENCRYPTION (JWE)

Testing JWE (iii)

- Authenticate with JWE(i):

POST | http://localhost:8080/HeloWorld/demo/hello/testJWE | Send

Params | Auth | Headers (9) | Body | Pre-req. | Tests | Settings | Cookies

Type: Digest Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

By default, Postman will extract values from the received response, add it to

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

Username: restUser2

Password: ••••••••••••••••

Show Password

- Paste the obtained token in a new header param

POST | http://localhost:8080/HeloWorld/demo/hello/testJWE | Send

Params | Auth | Headers (9) | Body | Pre-req. | Tests | Settings | Cookies

Headers (8 hidden)

	KEY	VALUE	DESCRIP	Bulk Edit	Presets
☑	token	eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q...			
	Key	Value	Description		

JSON WEB ENCRYPTION (JWE)

Testing JWE (iv)

- Authenticate with JWE(ii):

POST http://localhost:8080/HeloWorld/demo/hello/testJWE Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw Text

1 Guadalupe

Body Cookies Headers (5) Test Results 200 OK 14 ms 165 B Save Response

Pretty Raw Preview Visualize Text

1 Hello Guadalupe

Support Bibliography and References

- Amauri Valdés, Developers Corner. https://avaldes.com/category/java-development/jax-rs/
- Andrés Salazar C., René Enríquez. RESTFUL Java Web Services Security. https://www.safaribooksonline.com/library/view/restful-java-web/9781783980109/
- Bill Burke. RESTFUL Java with JAX-RS 2.0. https://www.safaribooksonline.com/library/view/restful-java-with/9781449361433/
- OWASP. REST Security Cheat Sheet. https://www.owasp.org/index.php/REST_Security_Cheat_Sheet#Introduction
- Guy Levin. Top 5 REST API Security Guidelines. https://dzone.com/articles/top-5-rest-api-security-guidelines
- Brian Campbell. Jose4j/JWT Examples. https://bitbucket.org/b_c/jose4j/wiki/JWT%20Examples