Basic Security in Distributed Applications

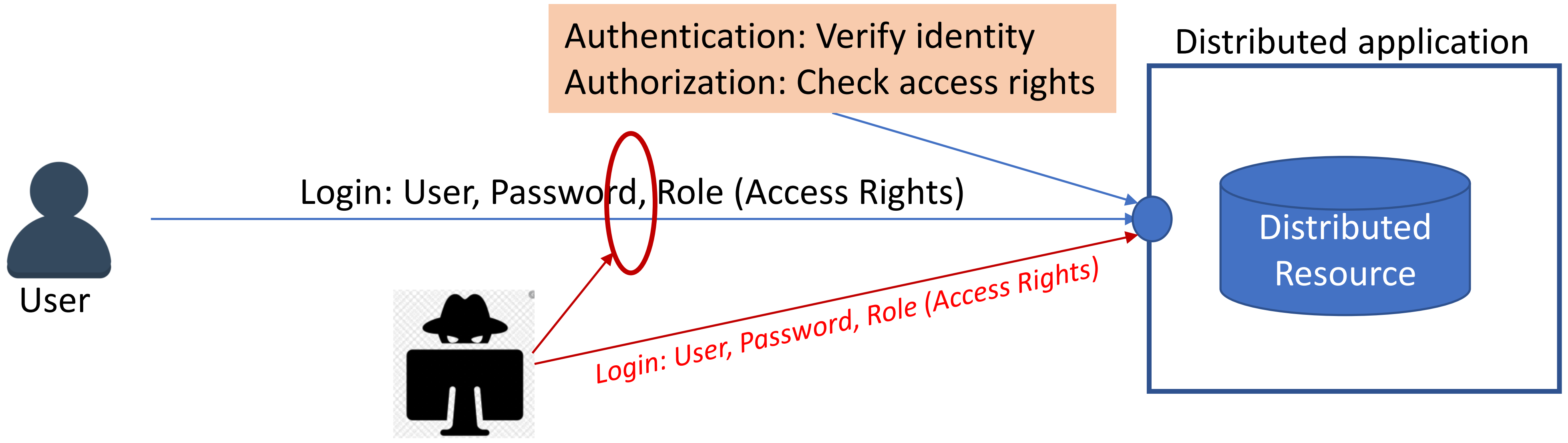
Security Designs for Microservices

2023

Contents

Application Security

- Distributed applications can be accessed remotely via network
- Access to resources must be restricted for different users
- Techniques for **authenticating** (verifying identity) and **authorization** (verifying if user has rights to access different resources) are developed
- However, attackers (Hackers) can impersonate real users. Thus, techniques to hide data from attackers are employed => use of **CRYPTOGRAPHY**


Authentication: Verify identity
Authorization: Check access rights
Distributed application
Login: User, Password, Role (Access Rights)
User
Distributed Resource
Login: User, Password, Role (Access Rights)

Hackers can eavesdrop to steal data and impersonate the real user

General Cryptography Concepts

- Fundamental component of any security solution
- Only method to ensure that the messages sent of the communication media cannot be understood by attackers, even if intercepted.
- Provides integrity and confidentiality protection
- Important for mechanisms of
  - Identity
  - Authenticity and
  - Non-repudiation
- Uses cryptographic methods with **symmetric** and **asymmetric** keys

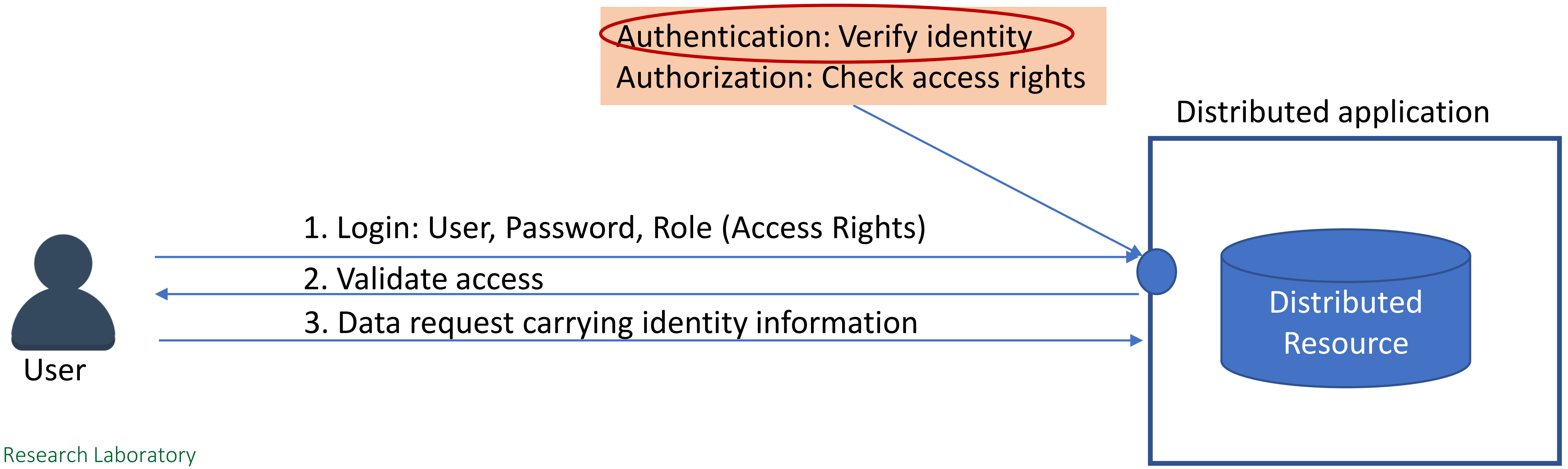
- Symmetric algorithms:
  - **E.g. Advanced Encryption Standard** (**AES**)
  - Problem with key-distribution

- Asymmetric algorithms:
  - Mainly used for symmetric algorithms key distribution
  - Diffie-Hellman
  - RSA
  - Elliptic Curve Cryptography

Application Security

Authentication

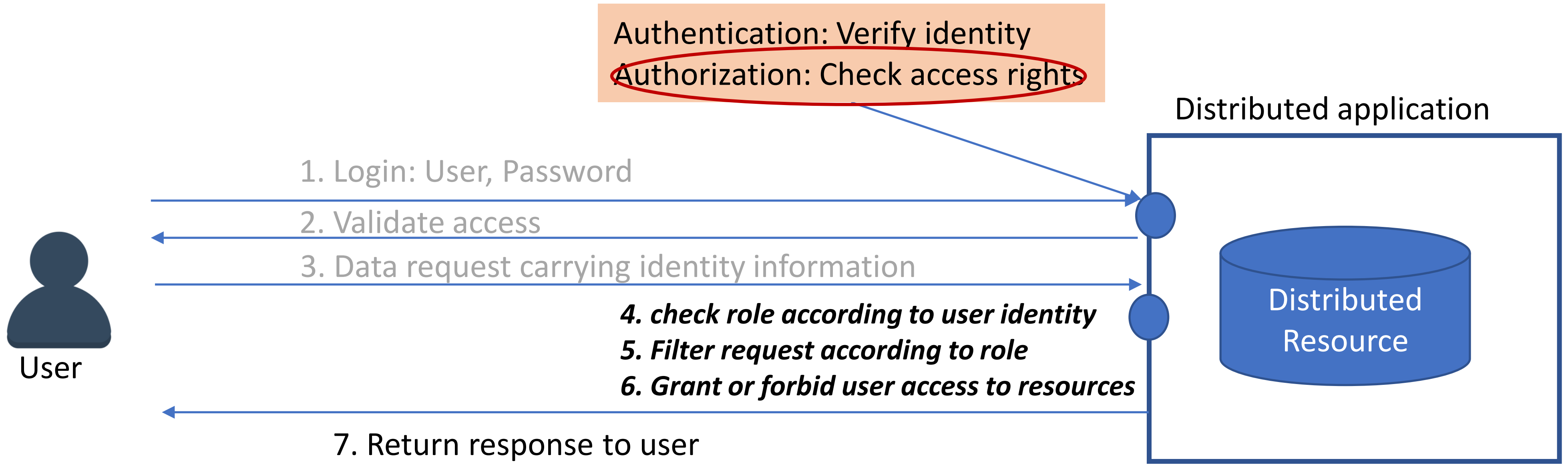
- Using a *Login* process
  - Standard use of a *username* and *password* to identify users
  - More complex forms can be employed (two-factor authentication, biometrics, etc.)
- User data should be sent to server to validate credentials
- All subsequent requests should carry some form of user authentication data to ensure that the same user is logged in
  - User data is saved in client applications (e.g. cookie, session or local storage)


Authentication: Verify identity
Authorization: Check access rights
Distributed application
1. Login: User, Password, Role (Access Rights)
2. Validate access
3. Data request carrying identity information
Distributed Resource
User

Application Security

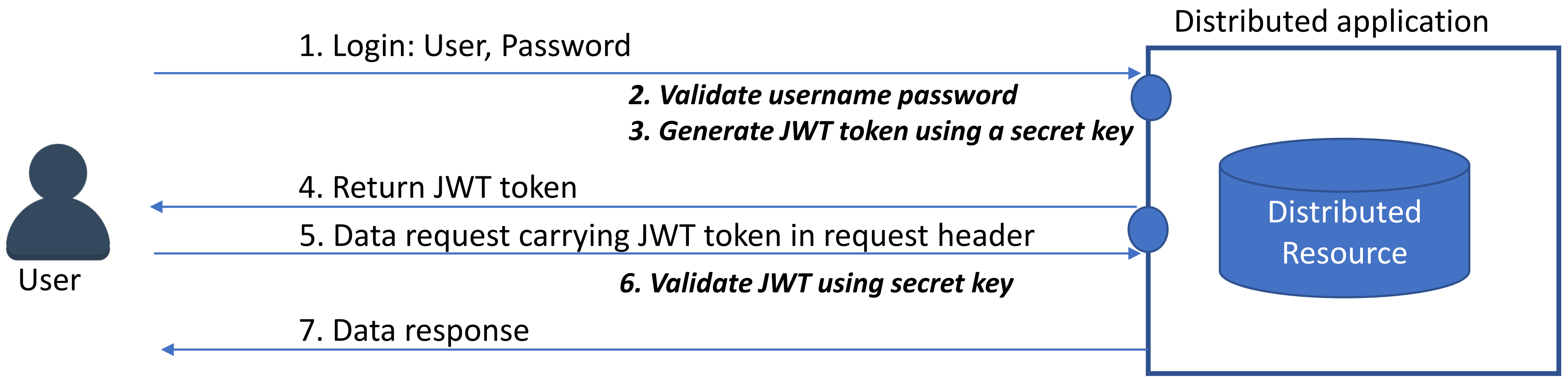
Authorization

- Using a set of *FILTERS according to access rights (or roles)* on the requests
    - Standard use of a *different application roles to encode the access rights*
- User access rights are created during register process
- After each authentication process (login), the user access rights are returned from the database and used at each subsequent request to allow user access to different resources
- User access rights should be stored only on server side, not to allow users to alter them
    - Can be saved in a session at the server side
    - Can be retrieved from DB at each request (too time consuming)


Authentication: Verify identity
Authorization: Check access rights
Distributed application
1. Login: User, Password
2. Validate access
3. Data request carrying identity information
4. check role according to user identity
5. Filter request according to role
6. Grant or forbid user access to resources
7. Return response to user
User
Distributed Resource

JWT (Json Web Token)

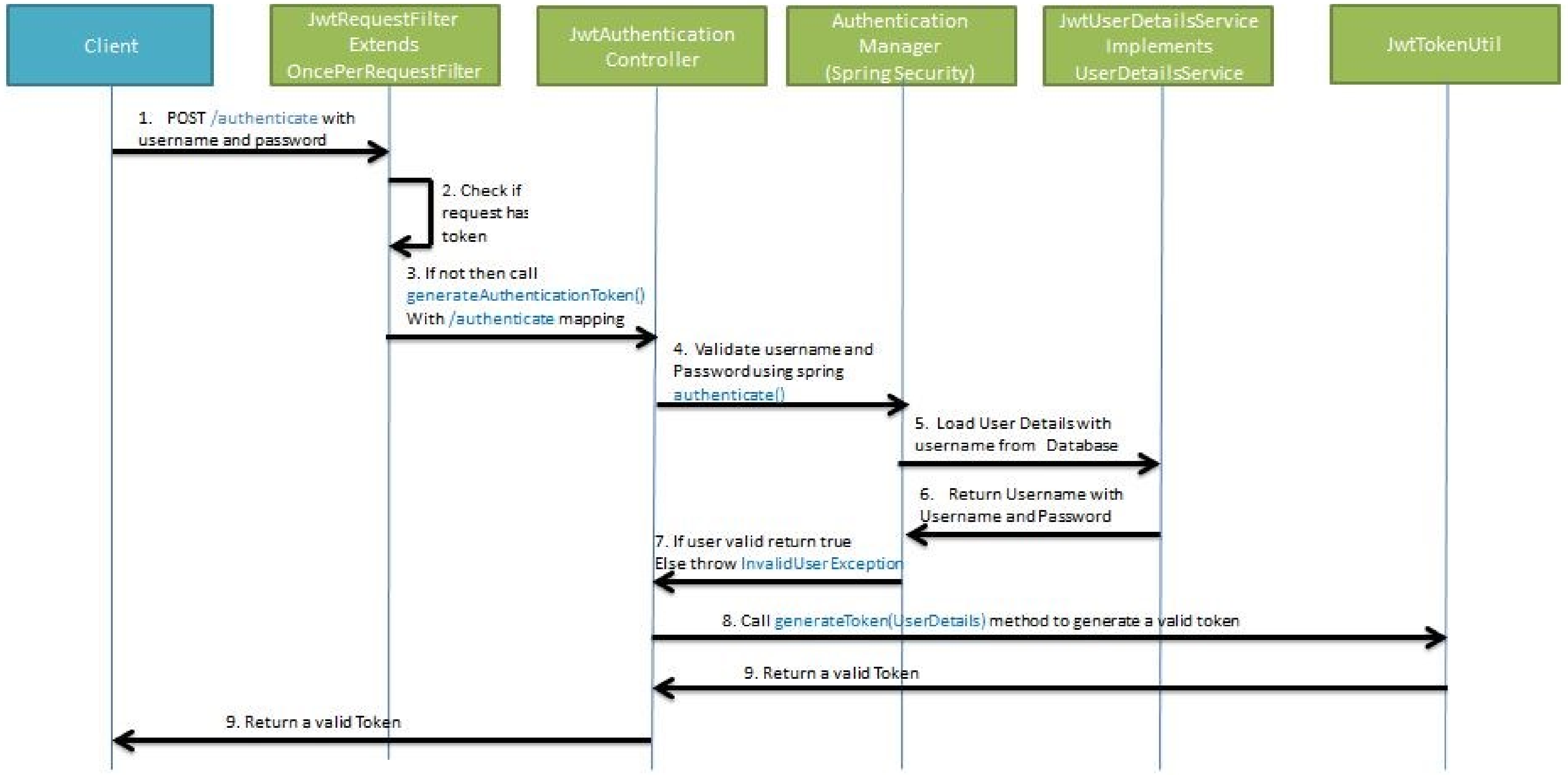
- Open standard (RFC 7519)
- Defines a compact format for transmitting information between clients and servers as JSON objects.
- Transmitted information can be verified because it is digitally signed.
- Reduce transmitted information in case of authentication (send JWT token instead of username and password)


Distributed application
1. Login: User, Password
2. Validate username password
3. Generate JWT token using a secret key
4. Return JWT token
5. Data request carrying JWT token in request header
6. Validate JWT using secret key
7. Data response
User
Distributed Resource

[Source: https://www.javainuse.com/spring/jwt]

Spring Security + JWT

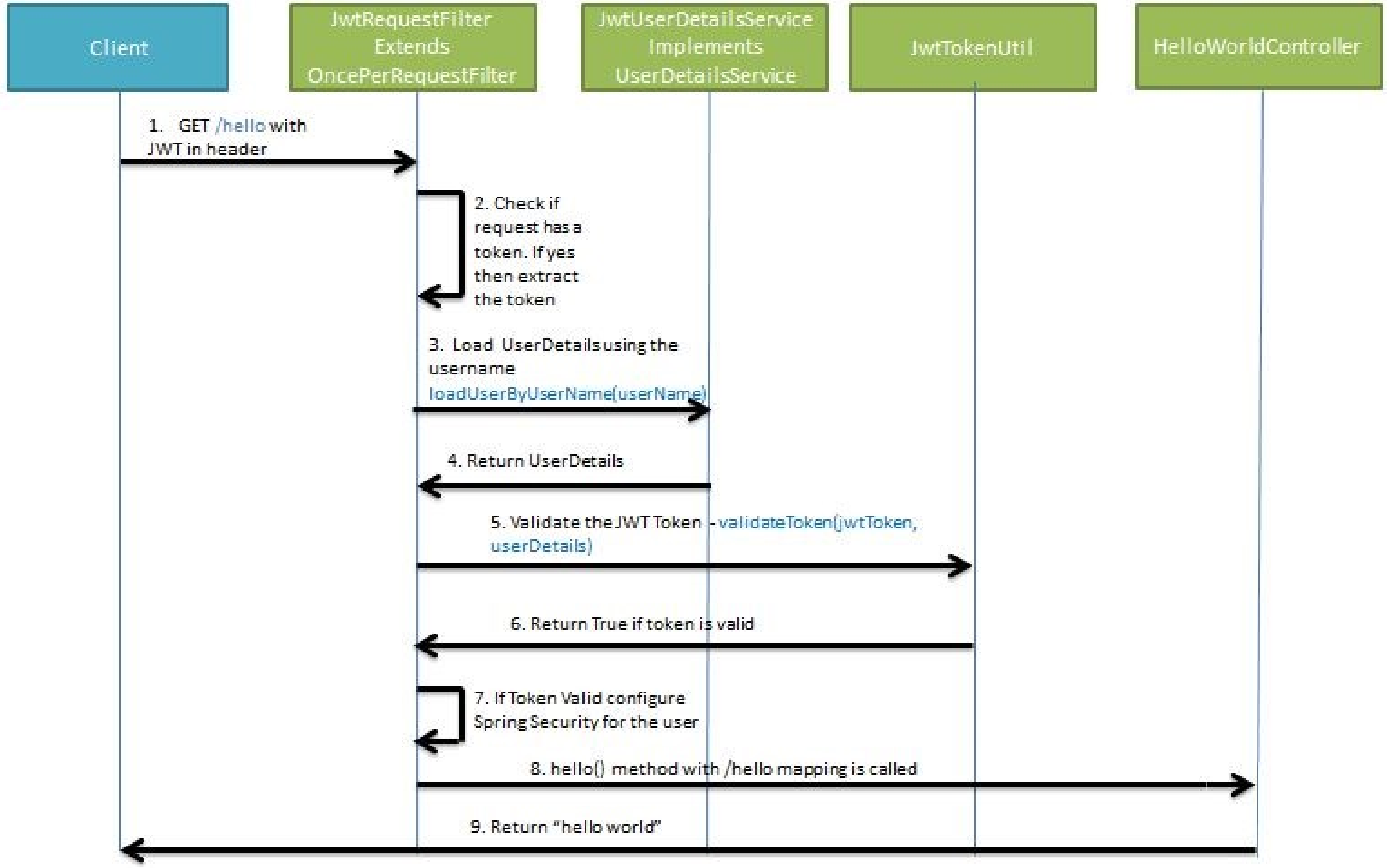
- Spring Security is a JAVA framework that provides authentication and authorization
- Develop a Spring application with a REST API
- Generate a JWT token at the first request



Client
JwtRequestFilter Extends OncePerRequestFilter
JwtAuthentication Controller
Authentication Manager (Spring Security)
JwtUserDetailsService Implements UserDetailsService
JwtTokenUtil
1. POST /authenticate with username and password
2. Check if request has token
3. If not then call generateAuthenticationToken() With /authenticate mapping
4. Validate username and Password using spring authenticate()
5. Load User Details with username from Database
6. Return Username with Username and Password
7. If user valid return true Else throw InvalidUser Exception
8. Call generateToken(UserDetails) method to generate a valid token
9. Return a valid Token
9. Return a valid Token

[Image Source: https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world]

Spring Security + JWT



Client
JwtRequestFilter Extends OncePerRequestFilter
JwtUserDetailsService Implements UserDetailsService
JwtTokenUtil
HelloWorldController
1. GET /hello with JWT in header
2. Check if request has a token. If yes then extract the token
3. Load UserDetails using the username loadUserByUserName(userName)
4. Return UserDetails
5. Validate the JWT Token - validateToken(jwtToken, userDetails)
6. Return True if token is valid
7. If Token Valid configure Spring Security for the user
8. hello() method with /hello mapping is called
9. Return "hello world"

- Use the token in all subsequent requests
- Furthermore, it is possible to use filters to grant access on endpoints according to user roles

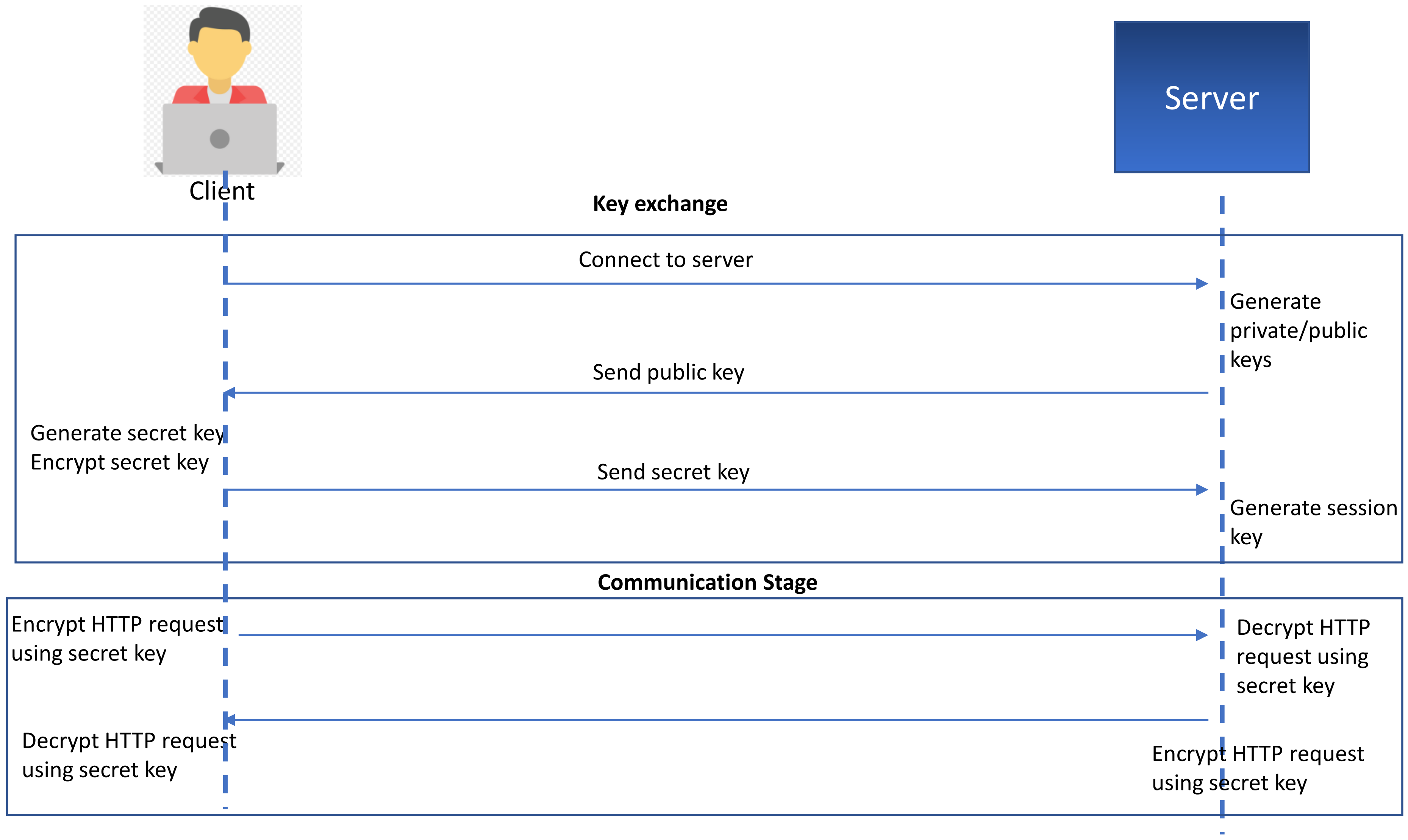
- ***Reference Tutorial: https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world***

[Image Source: https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world]

HTTPS Description

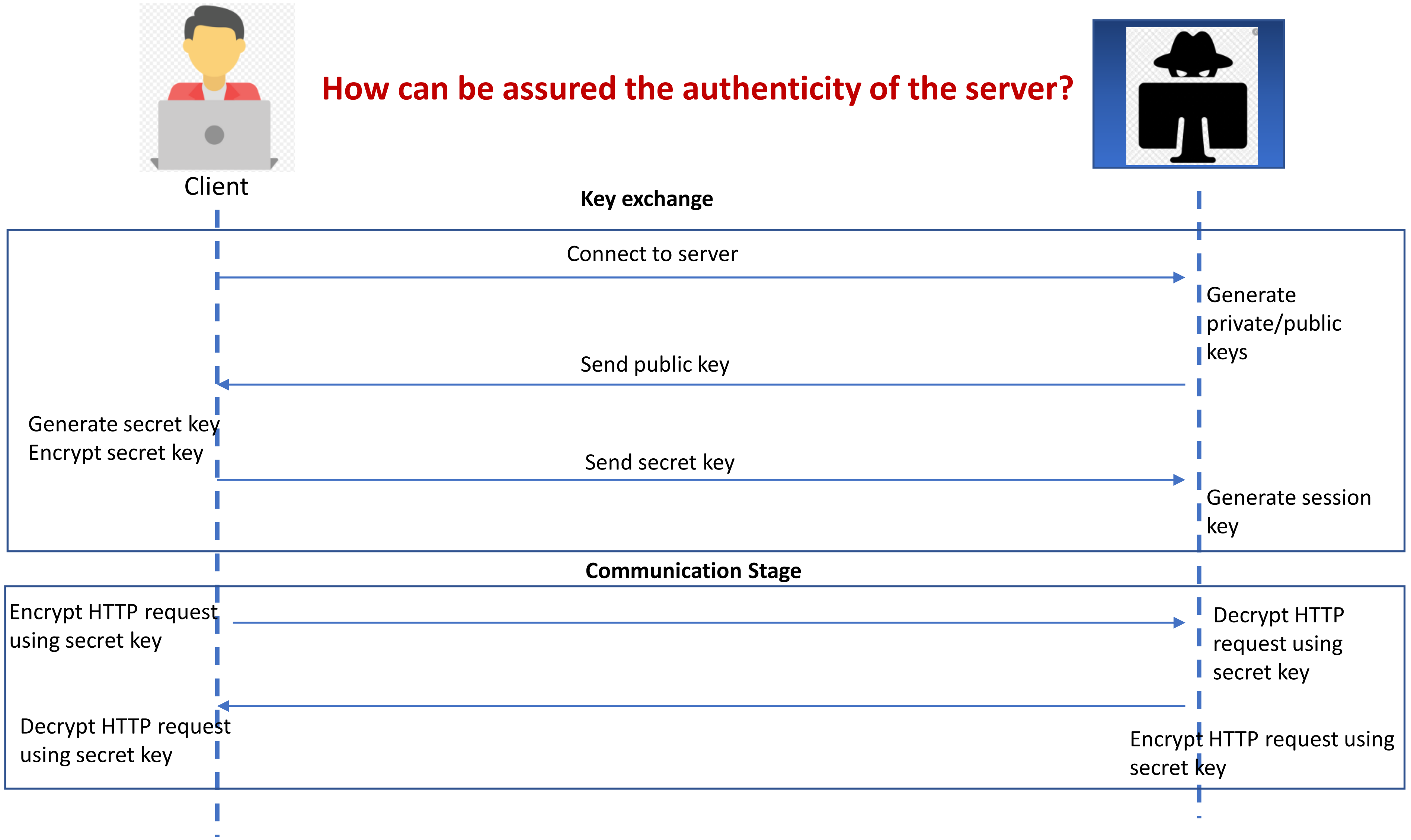
- HTTPS: **Hypertext Transfer Protocol Secure** (**HTTPS**)
- Secure communication over Transport Layer Security (TLS), or formerly Secure Sockets Layer (SSL)
- Port 443
- Relies on:
  - Asymmetric cryptography for key transmission
  - Symmetric cryptography for data transmission
  - Digital certificates for server authenticity
- Encrypt HTTP messages except destination base URL (IP: port)

HTTPS Connection Details

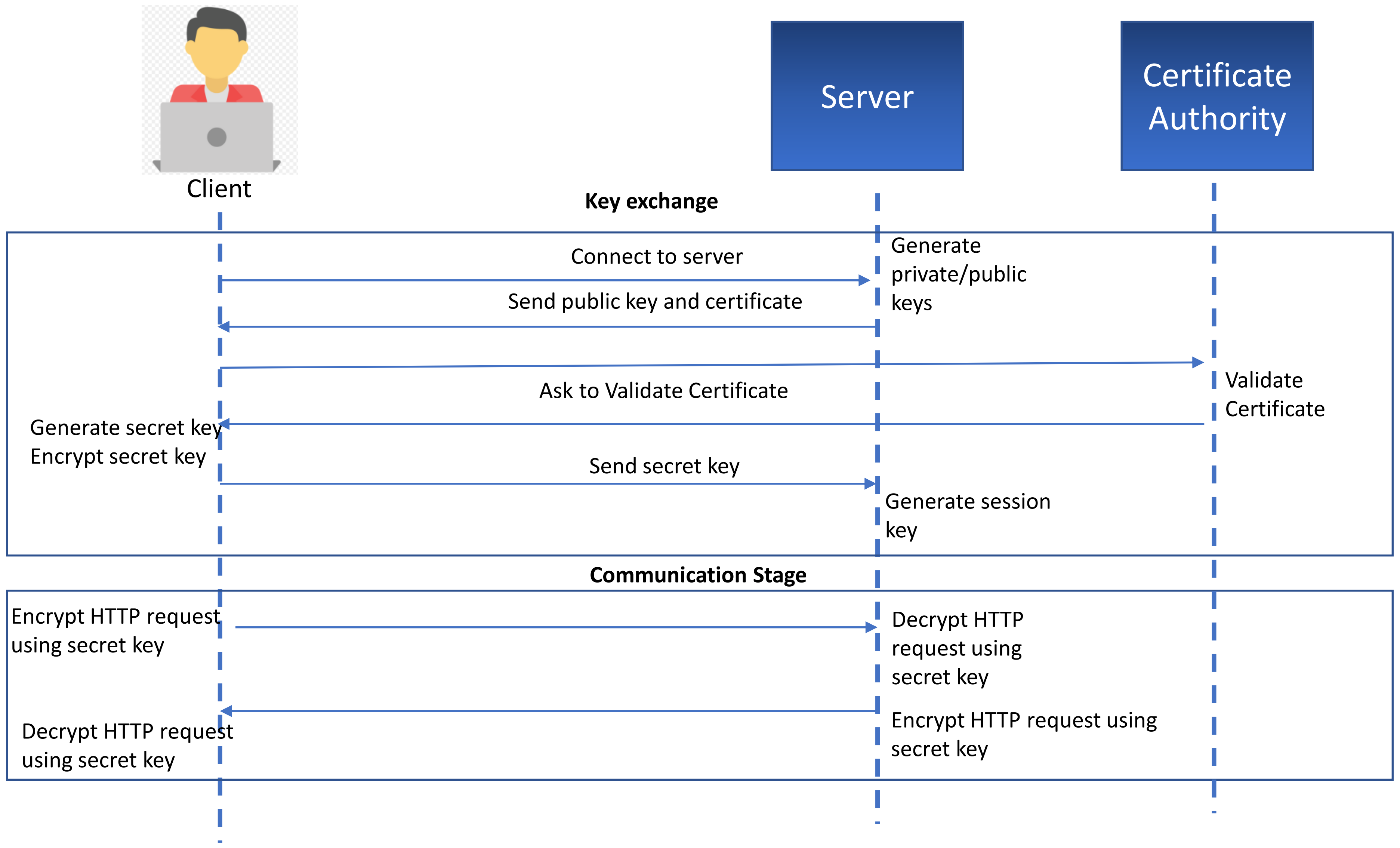

Client
Server
Key exchange
Connect to server
Generate private/public keys
Send public key
Generate secret key
Encrypt secret key
Send secret key
Generate session key
Communication Stage
Encrypt HTTP request using secret key
Decrypt HTTP request using secret key
Decrypt HTTP request using secret key
Encrypt HTTP request using secret key

HTTPS Connection Details

How can be assured the authenticity of the server?


Client
Key exchange
Connect to server
Generate private/public keys
Send public key
Generate secret key
Encrypt secret key
Send secret key
Generate session key
Communication Stage
Encrypt HTTP request using secret key
Decrypt HTTP request using secret key
Decrypt HTTP request using secret key
Encrypt HTTP request using secret key

HTTPS Connection Details



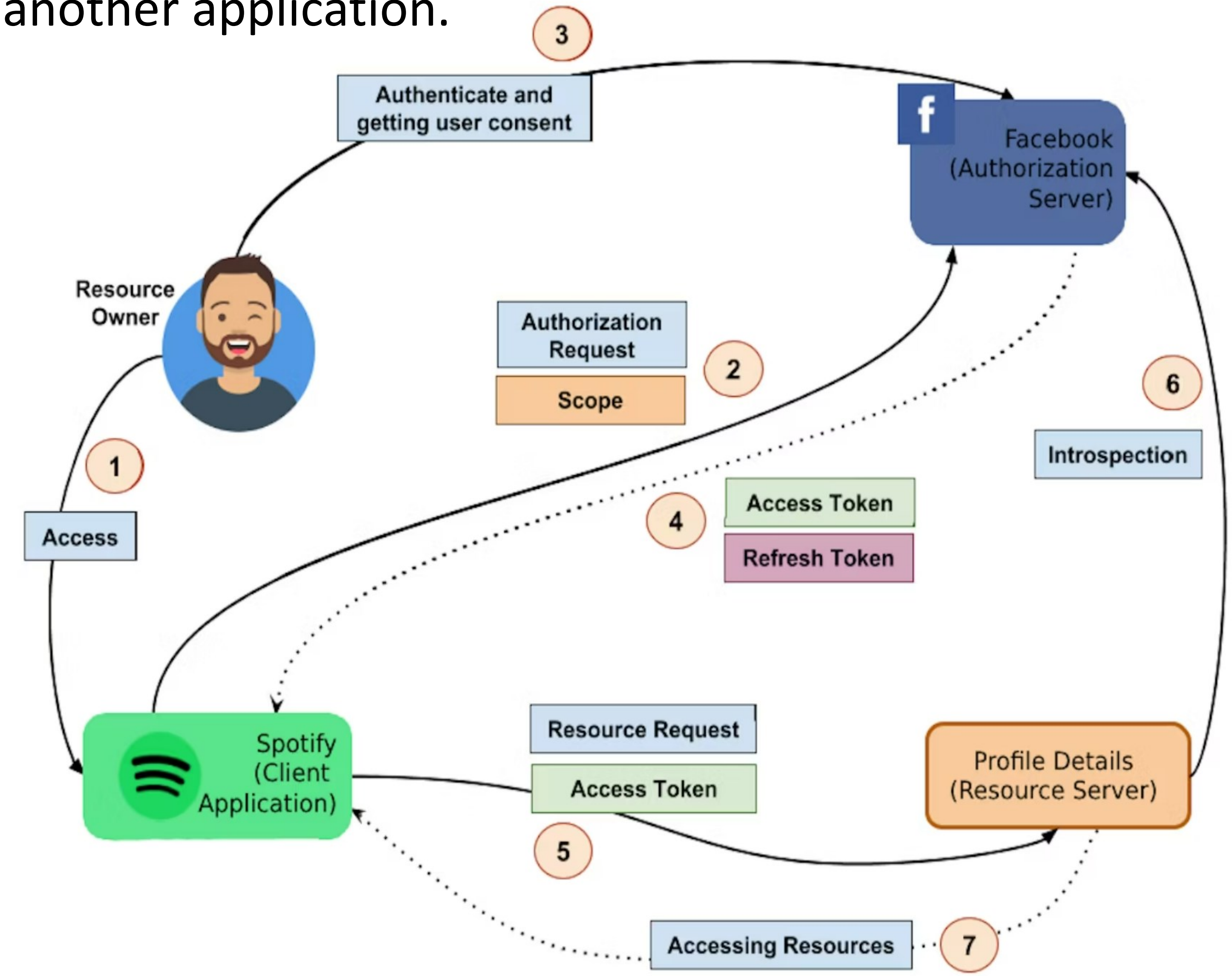
Client
Server
Certificate Authority
Key exchange
Connect to server
Generate private/public keys
Send public key and certificate
Ask to Validate Certificate
Validate Certificate
Generate secret key
Encrypt secret key
Send secret key
Generate session key
Communication Stage
Encrypt HTTP request using secret key
Decrypt HTTP request using secret key
Encrypt HTTP request using secret key
Decrypt HTTP request using secret key

HTTPS Practical Guide

- Configure Web Servers to use HTTPS instead of HTTP.
  - Requests will be handled on other port
  - Tomcat will use 8443 instead of 8080
- Generate a set of certificates to assure authenticity of server:
  - Generate own pair of public-private keys and add exception to browser
  - Buy a certificate from a CA

OAuth2 Protocol

- OAuth 2.0 is a security standard where one application gets permission to access data in another application.

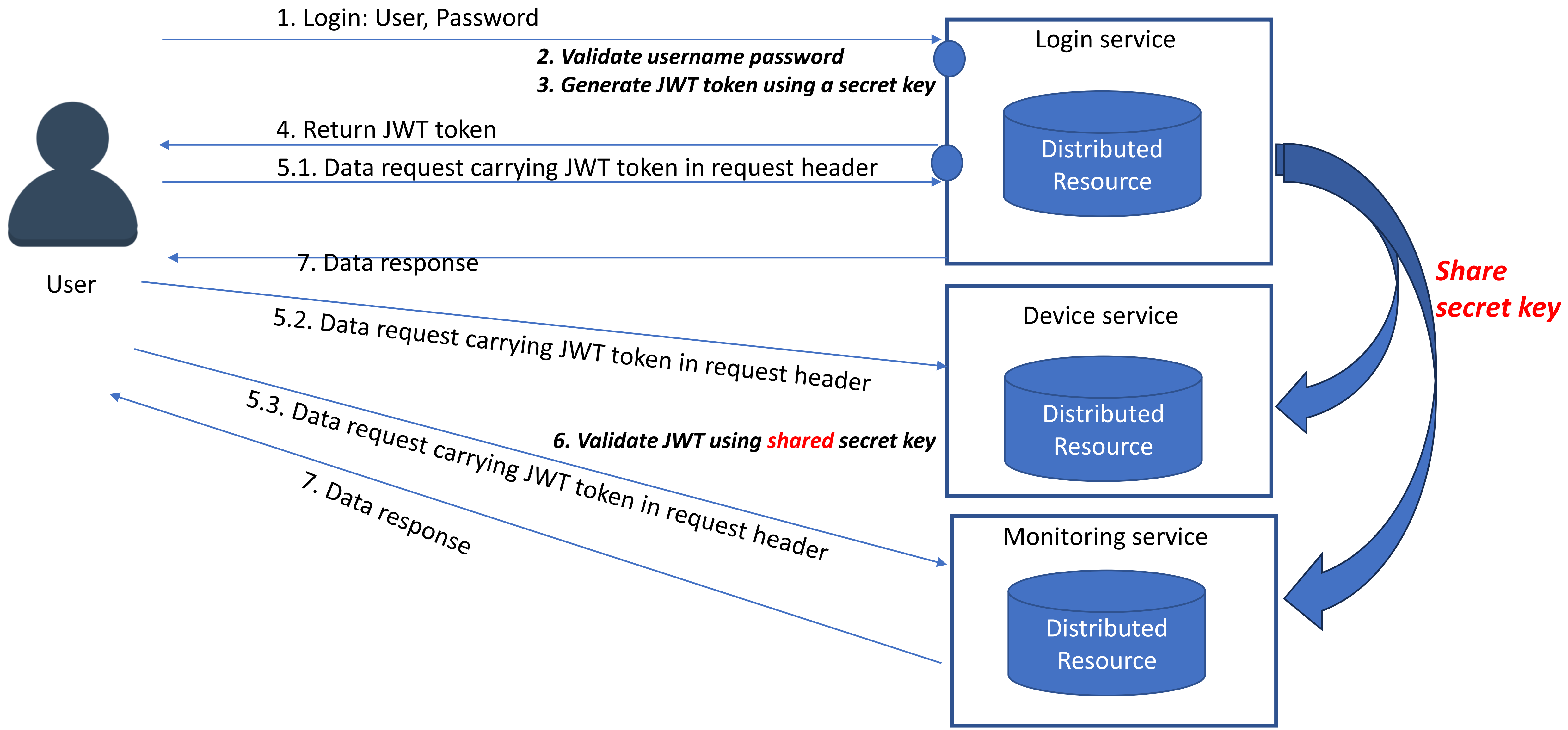


Sources: https://hackernoon.com/oauth-20-for-dummies

1. User requests access to Spotify (client application)
2. Client application makes request to Authorization Server (i.e. Facebook)
3. Authorization Server requests user credentials
4. Authorization Server sends Access Token and Refresh Token to client application.
5. Client application uses Access Token to access resources on Resource Server
6. Resource Server **verifies the access token** by sending it to Authorization Server
7. Client Application access resources on Resource Server using Access Token (that was already granted access)
8. When Access Token expires, it uses the Refresh to ask the Authorization Server for a new Access Token

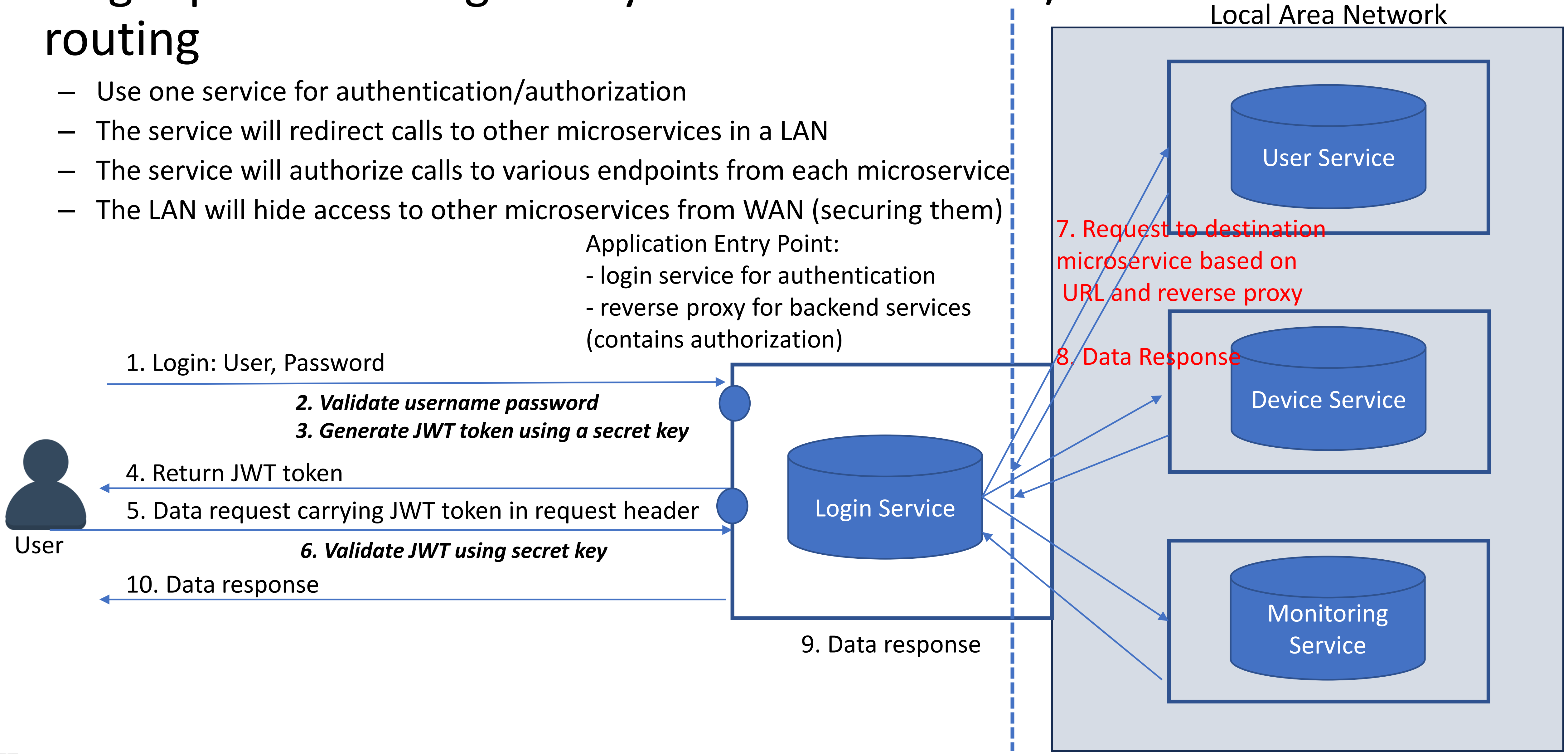
Security Design For Microservices

- JWT-based access with shared private key between resource servers


1. Login: User, Password
2. Validate username password
3. Generate JWT token using a secret key
4. Return JWT token
5.1. Data request carrying JWT token in request header
7. Data response
User
5.2. Data request carrying JWT token in request header
5.3. Data request carrying JWT token in request header
6. Validate JWT using shared secret key
7. Data response
Login service
Distributed Resource
Device service
Distributed Resource
Monitoring service
Distributed Resource
Share secret key

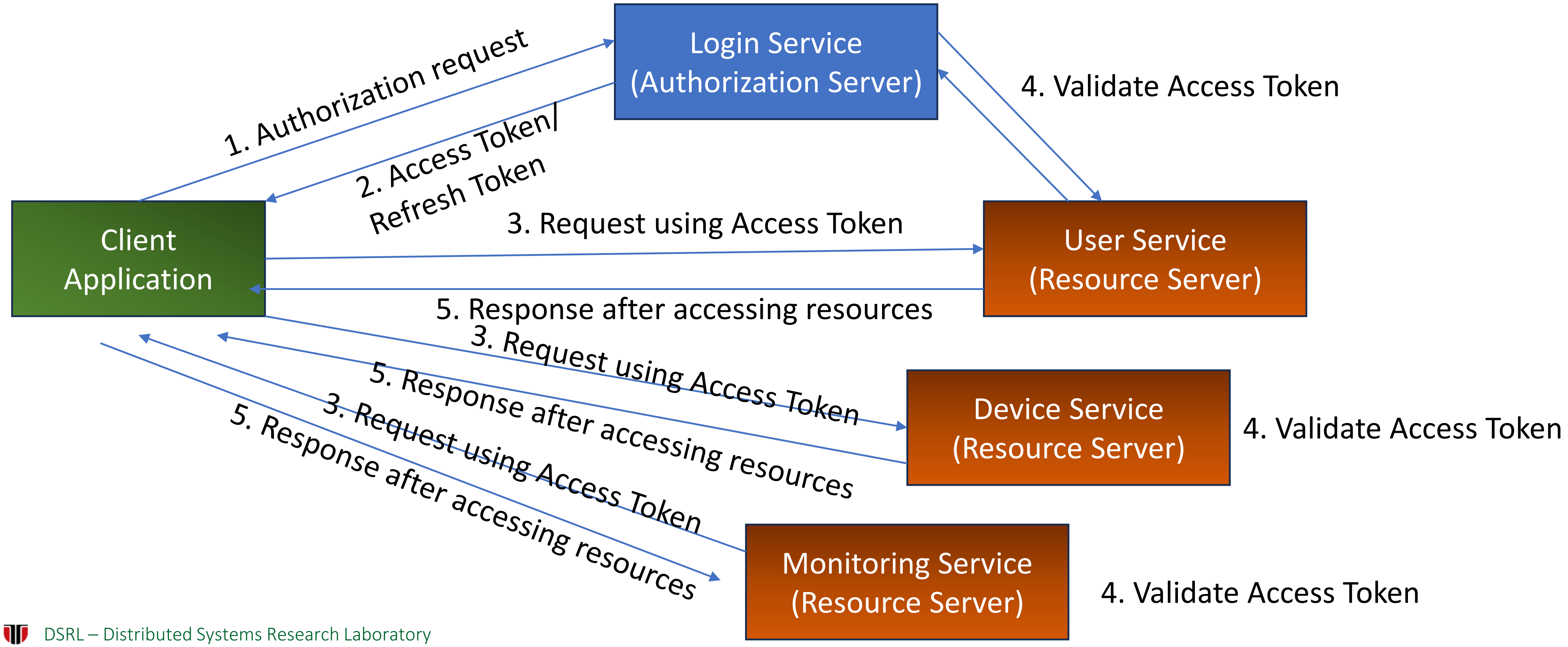
Security Design For Microservices

- Single-point access gateway for authentication/authorization and routing
  - Use one service for authentication/authorization
  - The service will redirect calls to other microservices in a LAN
  - The service will authorize calls to various endpoints from each microservice
  - The LAN will hide access to other microservices from WAN (securing them)


Local Area Network
Application Entry Point:
- login service for authentication
- reverse proxy for backend services
(contains authorization)
1. Login: User, Password
2. Validate username password
3. Generate JWT token using a secret key
4. Return JWT token
5. Data request carrying JWT token in request header
6. Validate JWT using secret key
10. Data response
9. Data response
User
Login Service
7. Request to destination microservice based on URL and reverse proxy
8. Data Response
User Service
Device Service
Monitoring Service

Security Design For Microservices

- Implement own OAuth 2 server
  - Implement OAuth2 Server in Spring
  - E.g. https://github.com/spring-projects/spring-security-samples/tree/main/servlet/spring-boot/java/oauth2/login


Login Service (Authorization Server)
1. Authorization request
2. Access Token/ Refresh Token
4. Validate Access Token
Client Application
3. Request using Access Token
User Service (Resource Server)
5. Response after accessing resources
3. Request using Access Token
5. Response after accessing resources
Device Service (Resource Server)
4. Validate Access Token
3. Request using Access Token
5. Response after accessing resources
Monitoring Service (Resource Server)
4. Validate Access Token

Conclusion

- Security is a Non-Functional requirement of Distributed Applications
- Security consists of Authorization, Authentication, Data Integrity, Non-Repudiation and Data Protection
- Cryptography is a powerful tool in assuring Application Security
- Authorization and Authentication can be assured through user identification (username and password) and roles.
- JWT is an Open Standard for the Authorization and Authentication process
- Data Integrity, Non-Repudiation and Data Protection can be obtained only using cryptography.
- HTTPS is a secure HTTP protocol that uses asymmetric cryptography for key exchange and symmetric cryptography for data exchange