In28Minutes

=============

1. **Csrf(Cross site request forgery)** – You are logged in to your bank website

# A cookie Cookie-A is saved into your web browser

1. You go to a malicious website without logging out
2. Malicious website executes a bank transfer without your knowledge using Cookie-A

# How can you protect from CSRF

1. Synchronizer token pattern

# A token created for each request

# to make an update(POST,PUT) you need a CSRF token from previous request

1. SameSite cookie(Set-Cookie: SameSite=Strict)

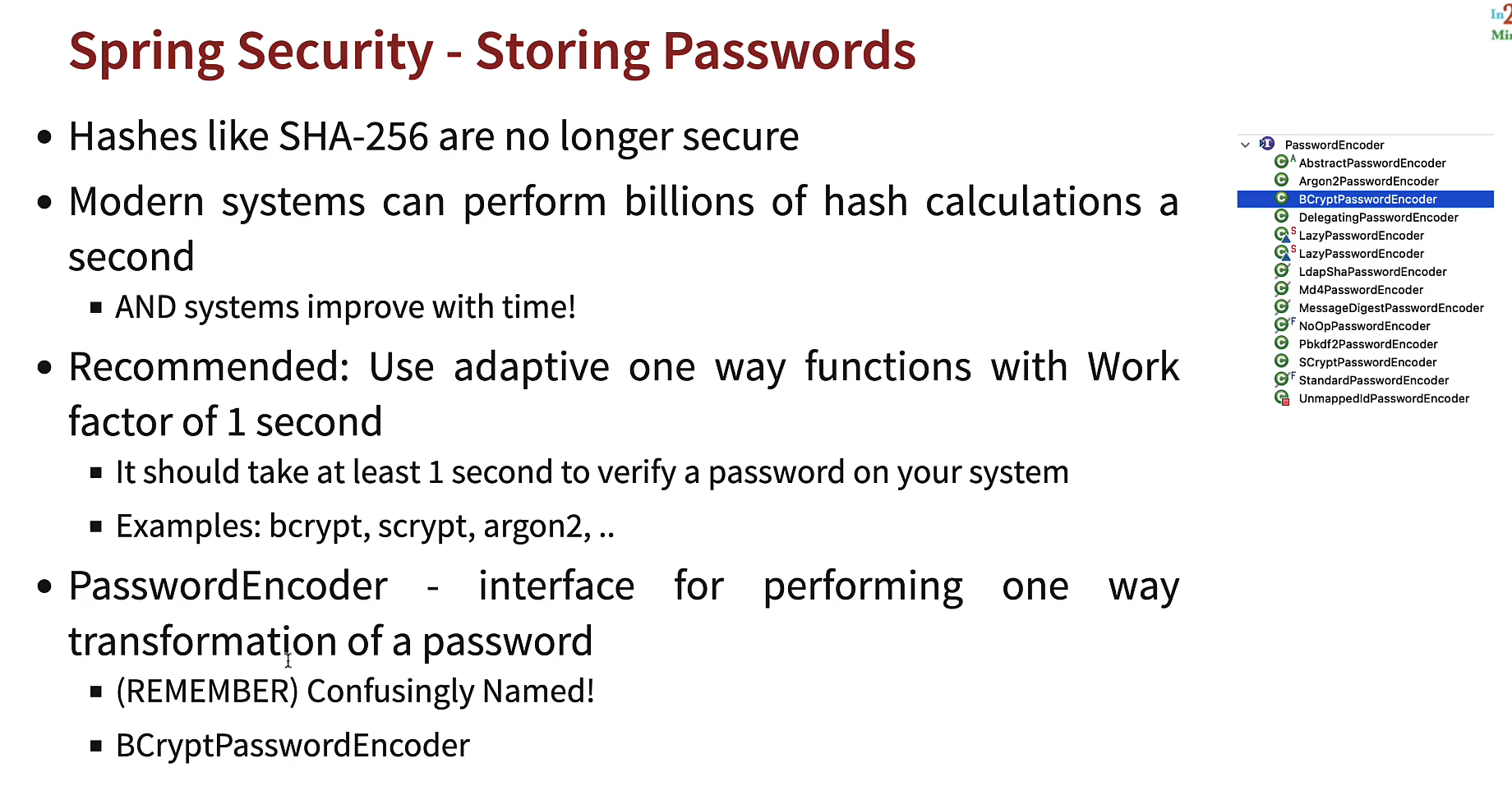
#application.properties

# server.servlet.session.cookie.same-site=strict

1. CSRF affects only if session maintains in web application , for stateless REST Api, we might want to disable CSRF.
2. To disable CSRF ->
3. Aaa
4. Aaa
5. HttpSecurity is the class which helps you to configure the filter chain. It allows configuring web based security for specific http requests.

Password Storing-> use **adaptive** **one-way functions** with **work factor 1 seconds**

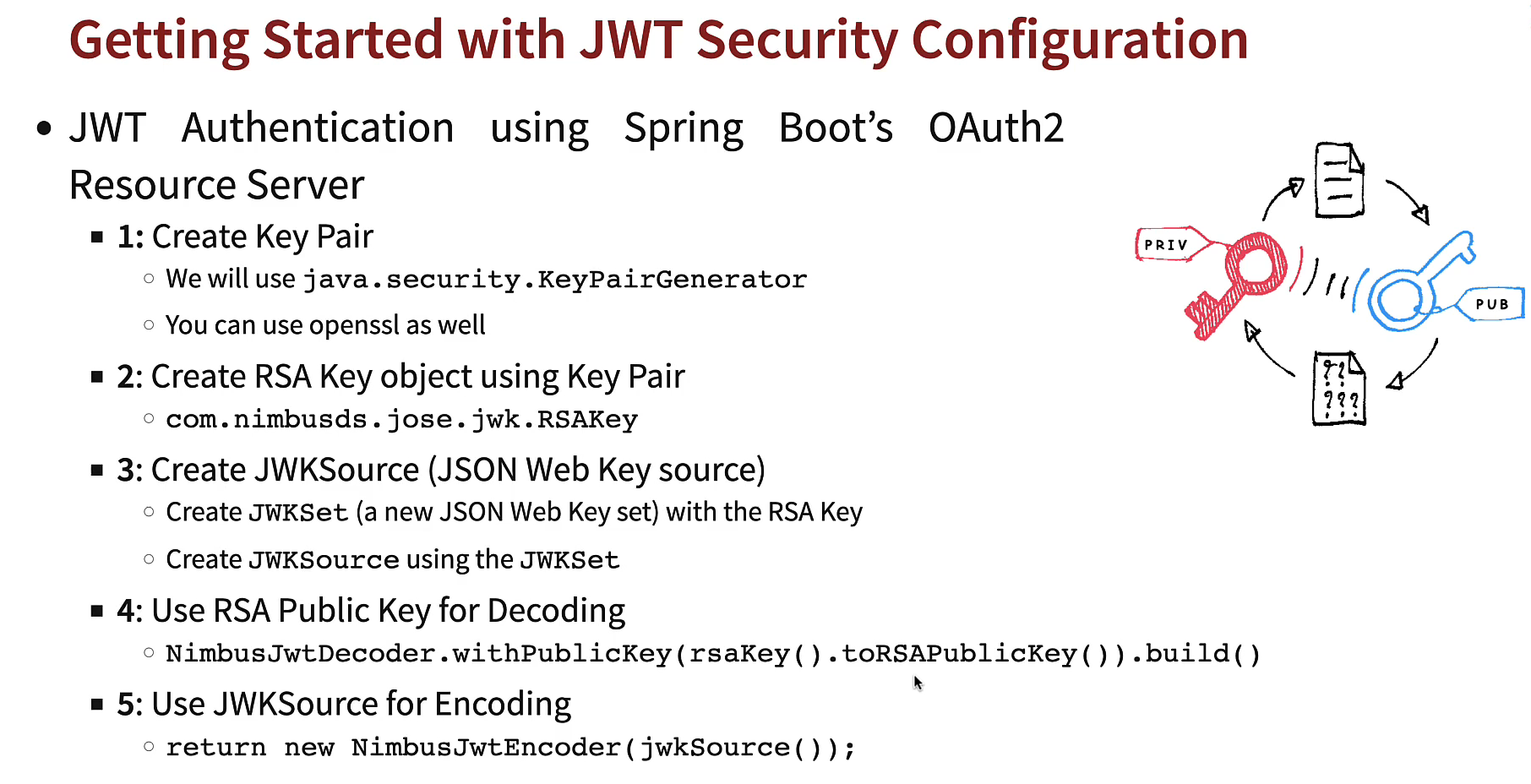
* **Adaptive** -> systems are improving with time, therefor Algorithms should be Adaptive, i.e. you should be able to configure something on algorithms which increases the **complexity ,** and that also increases the time it takes. Usually whenever we talk about computer algorithms , we would want them to be fast but in the case of password storing , if hashing is very very fast than somebody can do that hashing and try a brute force algorithms. And that’s why you don’t want your hashing algorithms to be too quick. Good examples of Adaptive one way functions are **BCrypt, SCrypt, Argon2** etc . in Spring Securityy there is a PasswordEncoder, which is used for performing one way transformation of a password. Recommended class is **BCryptPasswordEncoder.**
* **Work factor**-> it should take at least 1 second to verify/validate the password
* **One way**-> after hashing password would be converted into hash, but its reverse is not possible i.e. hash can not be converted back into plain text form again.
* **Biggest challenge is moden systems are very very powerful, they can perform huge number of hash calculations in a second and these systems are improving with the time. And that’s why its recommended to use adaptive one-way hash functions when it comes to storing your password.**



**JWT Security Configuration ->**

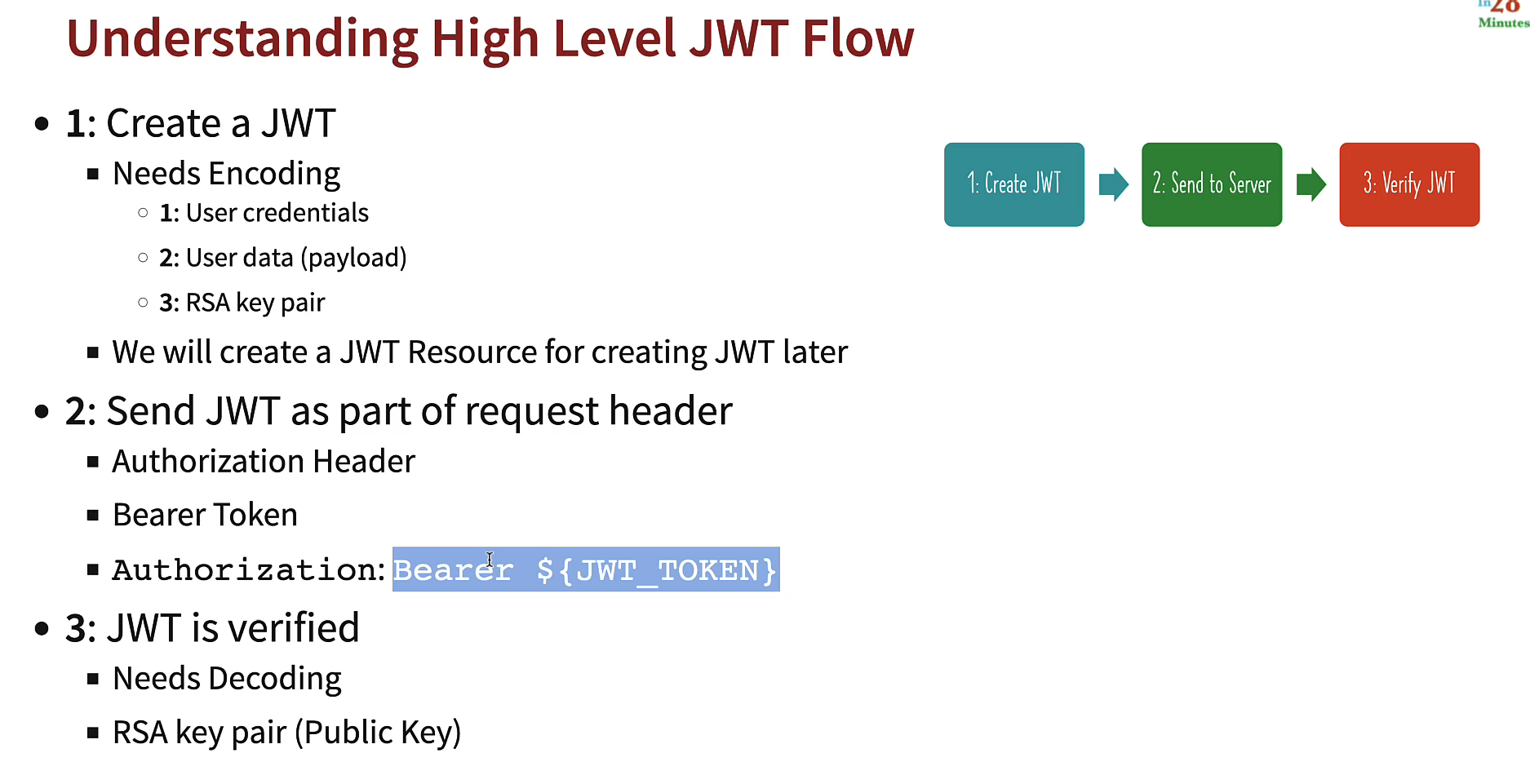
We will use Spring boot’s OAuth2 Resource Server for enabling JWT configuration

There are number of steps in enabling JWT authentication



Steps for JWT Configuration->

1. Enable OAuth2 Resource Server of Spring Boot
2. Resource server get this token

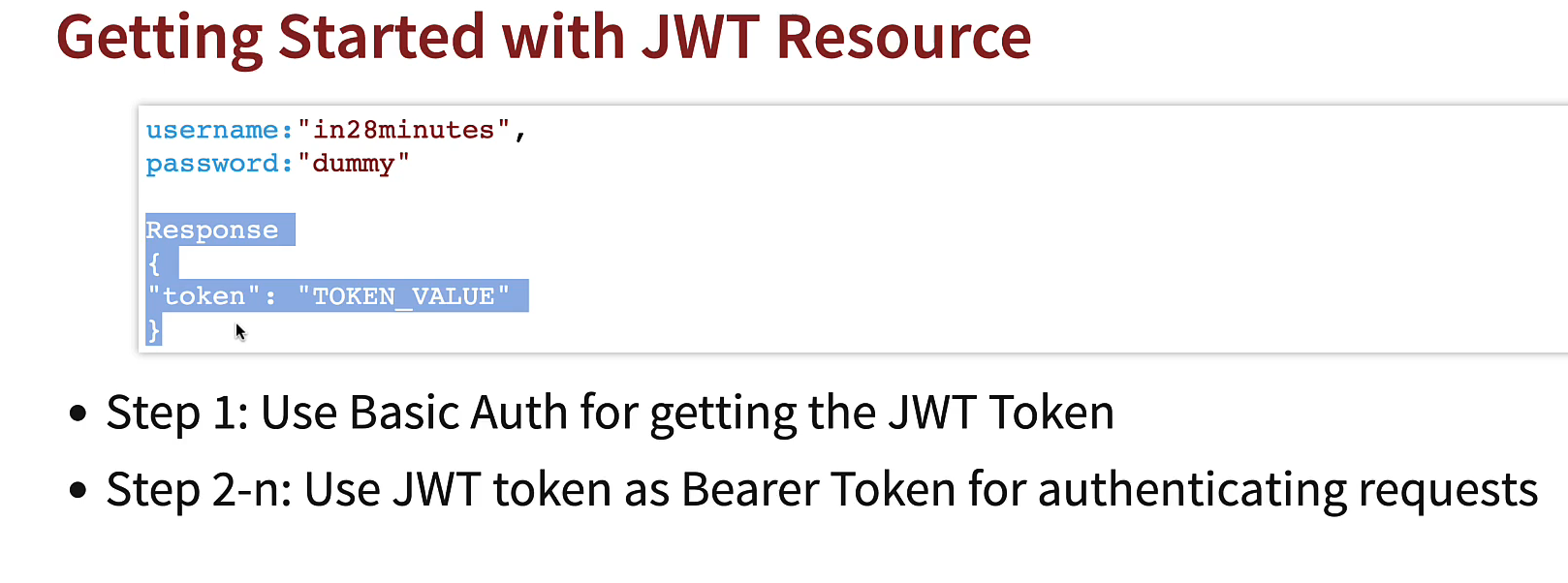


securityFilterChain() method would authenticate the request, this would verify the JWT

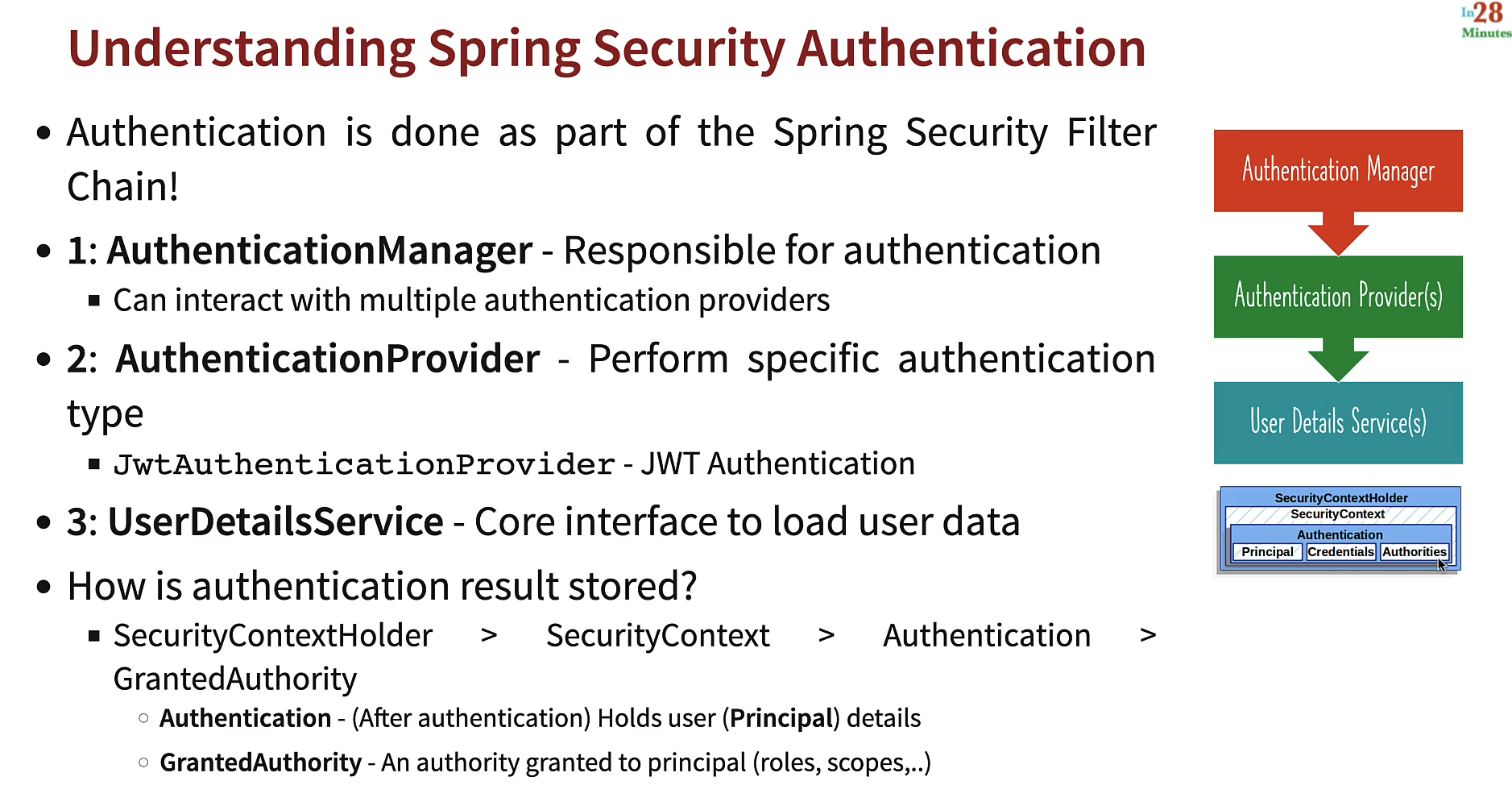
but if i would want to send the request to REST Api , i would first need to create the JWT.

To create JWT, we need to first create JWTResource

Whenever User wants to talk to REST Api, he would need to create a JWT token by sending a basic authentication request with username and password to the **JWTReSource, t**he response from JWTResource is JwtToken.

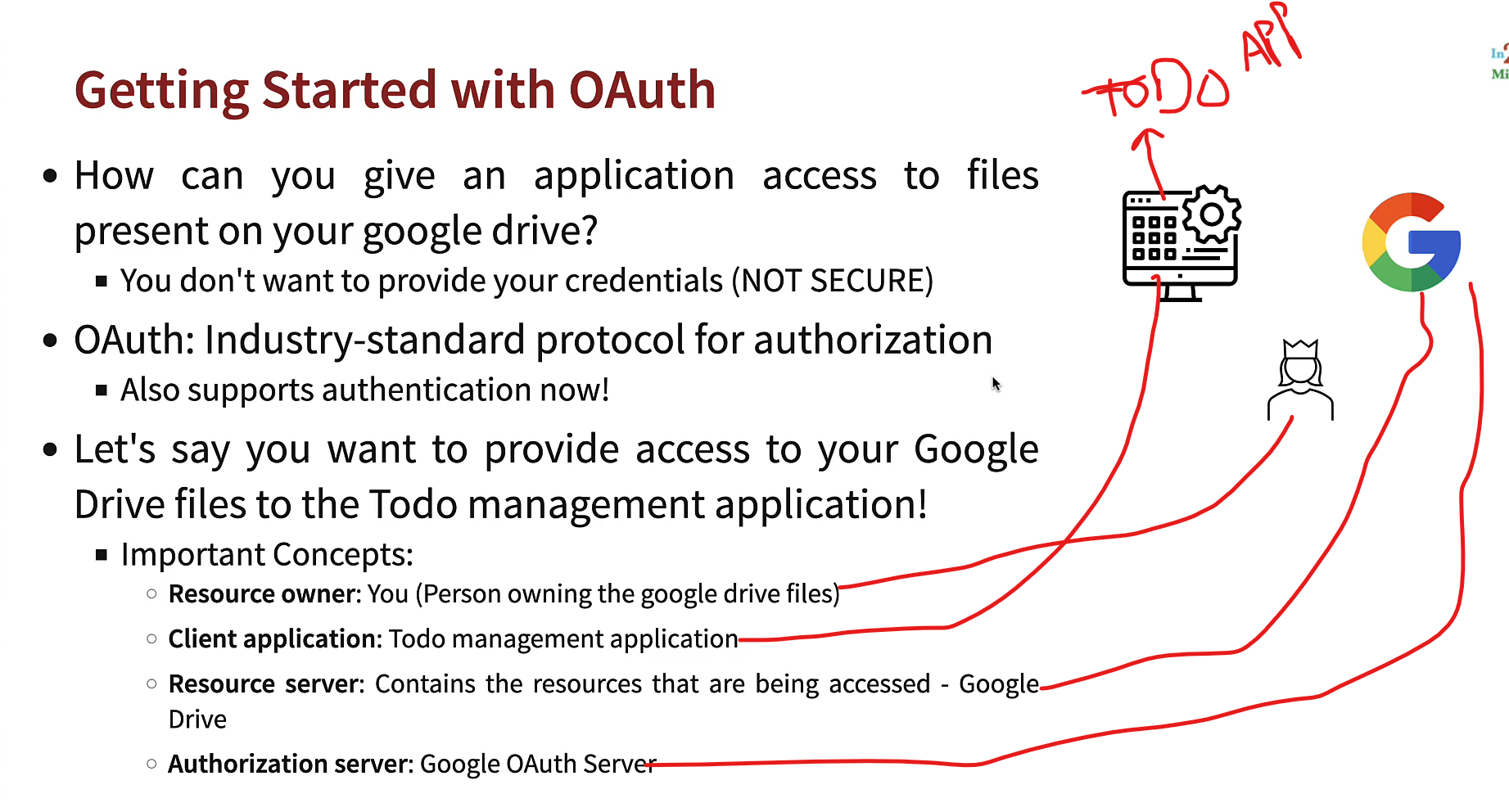
****

**Authentication Flow in Spring Security 🡪**

****

**AuthenticationManager** is main interface , it interacts with multiple AuthenticationProviders and UserDetailsService. UserDetailsService interacts with database or LDAP and get the user details and pass to AuthenticationProviders , which validate and check against the credentials and if successful than it would populate “Principal” and “Authorities” into “Authentication”.

**OAuth ->**

****

**Steps to avoid JWT token being stolen->**

1. Don’t Exposed JWT Token
2. Ephemeral solution- Log in at each application refresh
3. Use httpOnly cookie – JWT should be store into httpOnly cookie , httpOnly cookie is the cookie which can not be accessed by java script application.
4. Use HTTPS to encrypt data between server and client
5. Token encryption- encrypt JWT Payload by using Json Web Encryption (JWE
6. Token expiry- set a short expiration time for jwt tokens
7. Secure cookies- set the secure attribute on cookies to ensure they can sent over HTTPS only
8. Limit token scope- include only necessary information in JWT payload and avoid storing sensitive details like password etc
9. Implement Multi-factor-Authentication (MFA) to add additional security layer except JWT
10. Regular Security Audit- regular security audit like how JWT tokens are handled to identify and address any vulnerabilities

EazyBytes ->

Spring Security Flow Diagram->

