**Spring Security**

**Authentication**: It is about who tou are in the context of application.

E.g: User🡪sign in🡪application

Types: Http based, form based, certificate, tokens

**Authorization**: Ok you are authenticated but do you have permission to this task.

e.g: Application🡪admin section, user section , sales section(which section authorization)

Types:-

1.Privileges(Authorities):READ\_PROFILE,EDIT\_PROFILE,DELETE\_PROFILE,ACCESS\_PUBLIC\_API

It is granular(small pieces of permission)

2.Roles: ROLE\_ADMMIN,ROLE\_USER,ROLE\_SALES,ROLE\_MANAGEMENT

More bigger

**HTTP BASIC AUTHENTICATION**

Basic authentication is process for browser to request a username and password when making a request in order to authentify the user.

*GET /home(base 64)*

*Authorization: Basic xyzxyzxyzxyzxyz*

*200:ok*

*401:unauthorized*

Logic:

*Step 1: enter the username and password in the browser popup(below things are behind the scenes)*

*Step 2: userid and password are combined like below*

*Userid:password*

*Step 3: Encode it by Base 64*

*Step 4: Transmit it in HTTP header*

*Authoriztion: Basic kkkhgvddvjdvjdb*

**Important Points:-**

🡪It is very simple. It does not requires cookies, session identifiers or login page.

🡪Transmitted credentials are not encrypted. They are encoded in base 64 but are not encrypted or hashed.

🡪It is used in conjuction with HTTPS to provide confidentiality.

🡪No method to instruct the logout

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

//"ROLE\_ADMIN" added in authority as authority takes precedence ovver roles

auth.inMemoryAuthentication().withUser("admin").password(passwordEncoder().encode("admin123")).roles("ADMIN").authorities("ACCESS\_TEST1","ACCESS\_TEST2","ROLE\_ADMIN")

.and().withUser("kundan").password(passwordEncoder().encode("@kundan621Kk")).roles("USER")

.and().withUser("kundan1").password(passwordEncoder().encode("@kundan621Kk1")).roles("MANAGER");

}

//note in above case authority is bigger than role, if you want to use role in configure below then role also needs to be added in the authority above

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.antMatchers("/index.html").permitAll()

.antMatchers("profile/\*\*").authenticated()

.antMatchers("/admin/\*\*").hasRole("ADMIN") //role is admin so "ROLE\_ADMIN" added in authority

.antMatchers("/management/\*\*").hasAnyRole("MANAGER","ADMIN")

.antMatchers("/api/public/\*\*").hasAuthority("ACCESS\_TEST1").and().httpBasic();

}

@Bean

PasswordEncoder passwordEncoder(){

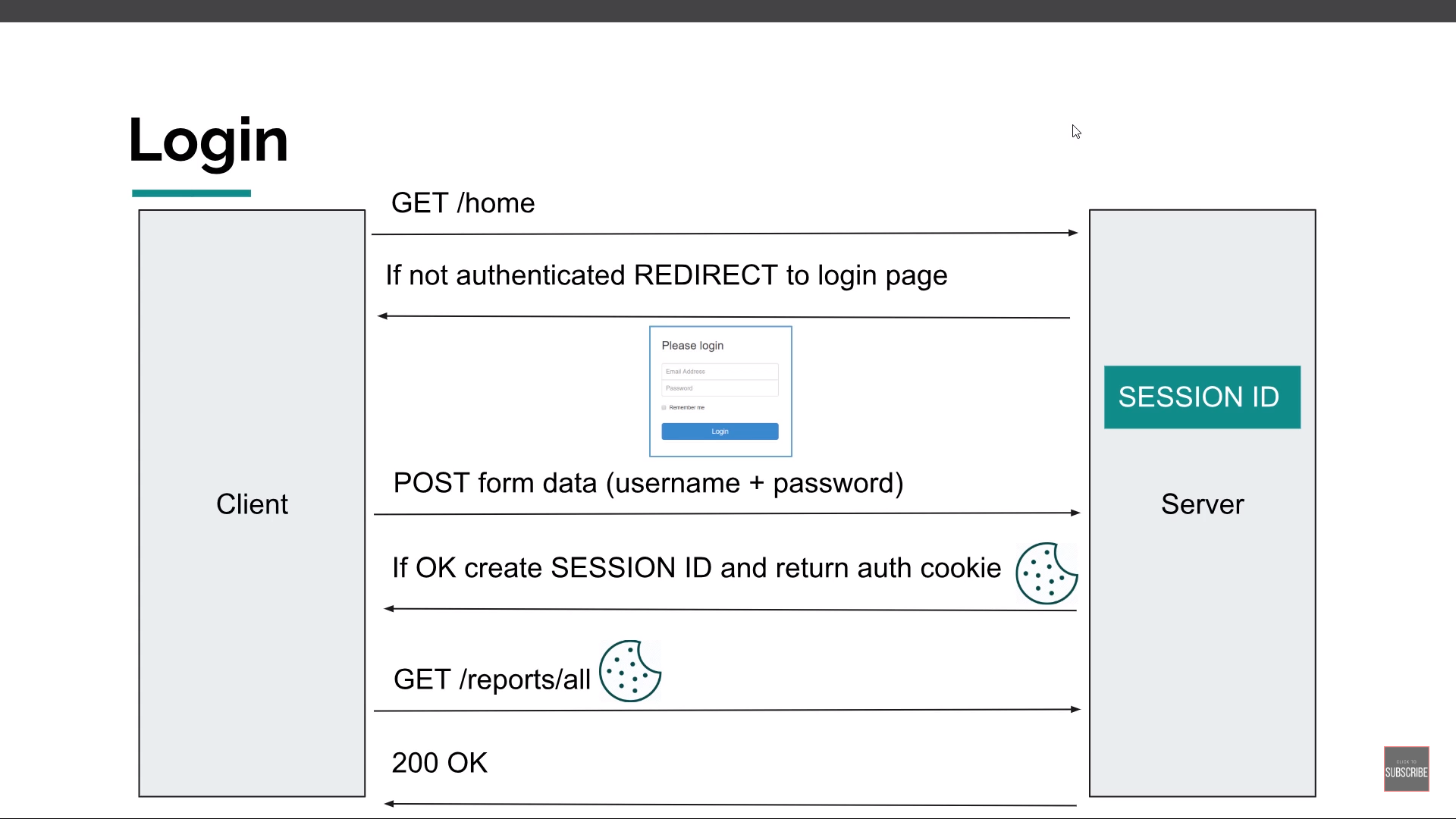
**return** **new** BCryptPasswordEncoder();

}

}

**Form Based Authentication**

Process of authenticating the user by presenting a custom html page that will collect the credentials and by directing the authentication responsibility to the web application that collect the form data



Application is responsible for dealing with form data.

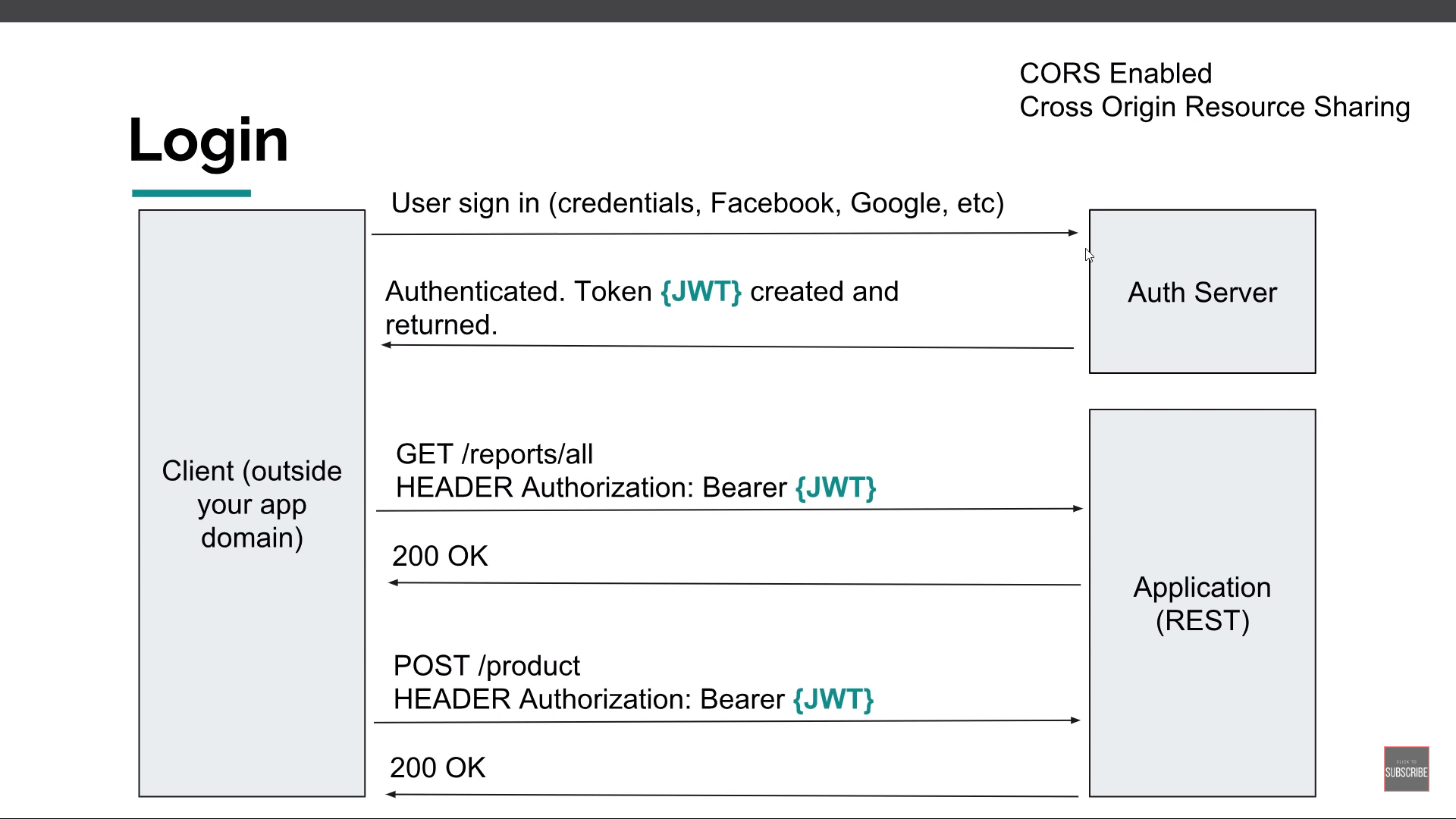
It is most widespread form of authentication.

User credential are conveyed in clear to web app so use SSL to keep them safe in transit.

Not suitable for public REST endpoints. Only self contained(eg website)

**JWT(json web token) based Authentication**

Jwt is a compact and safe way to transmit data between two parties. Information can be trusted as it is digitally signed.



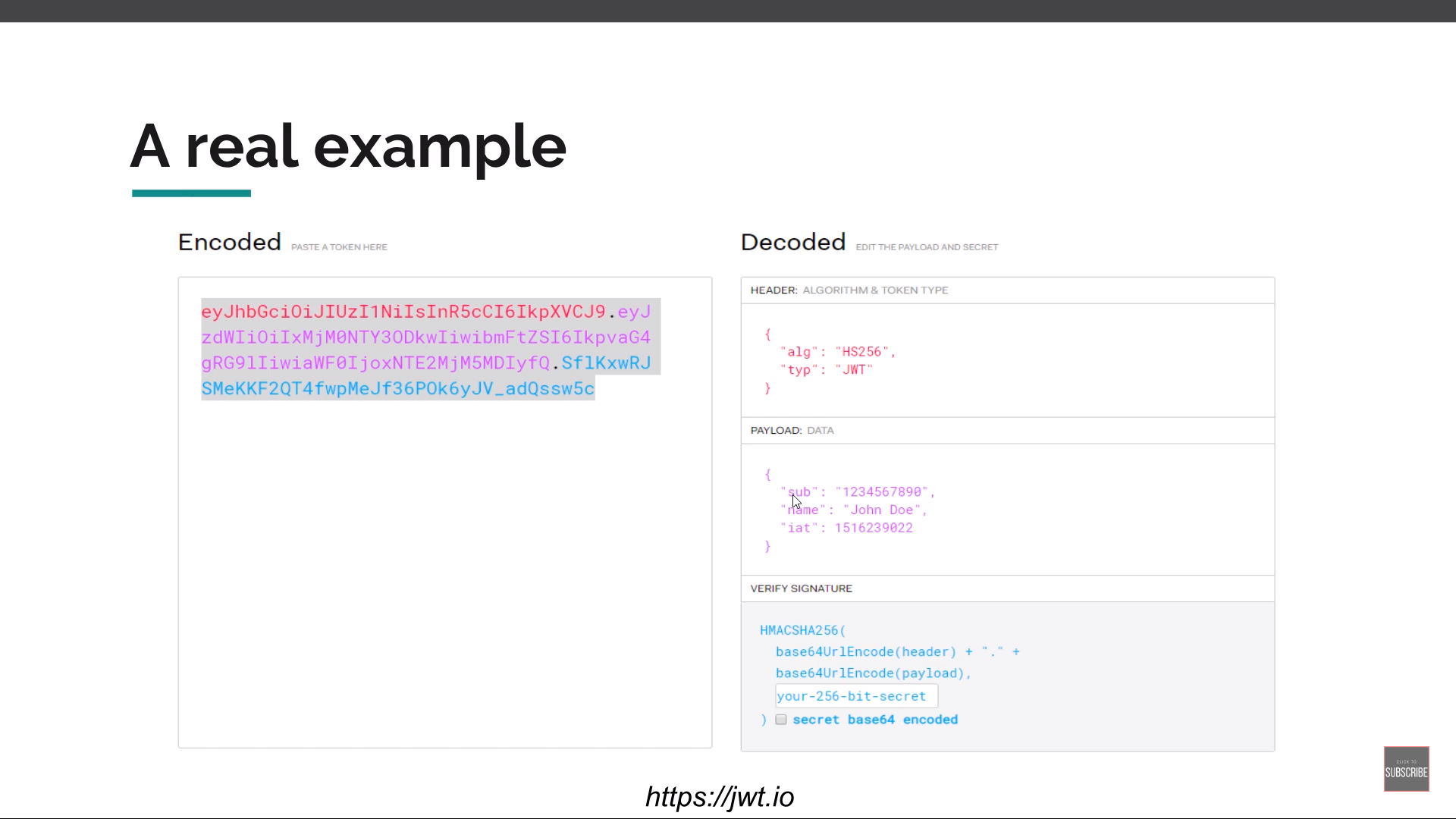
**Jwt structure**

Consists of three parts separated by :

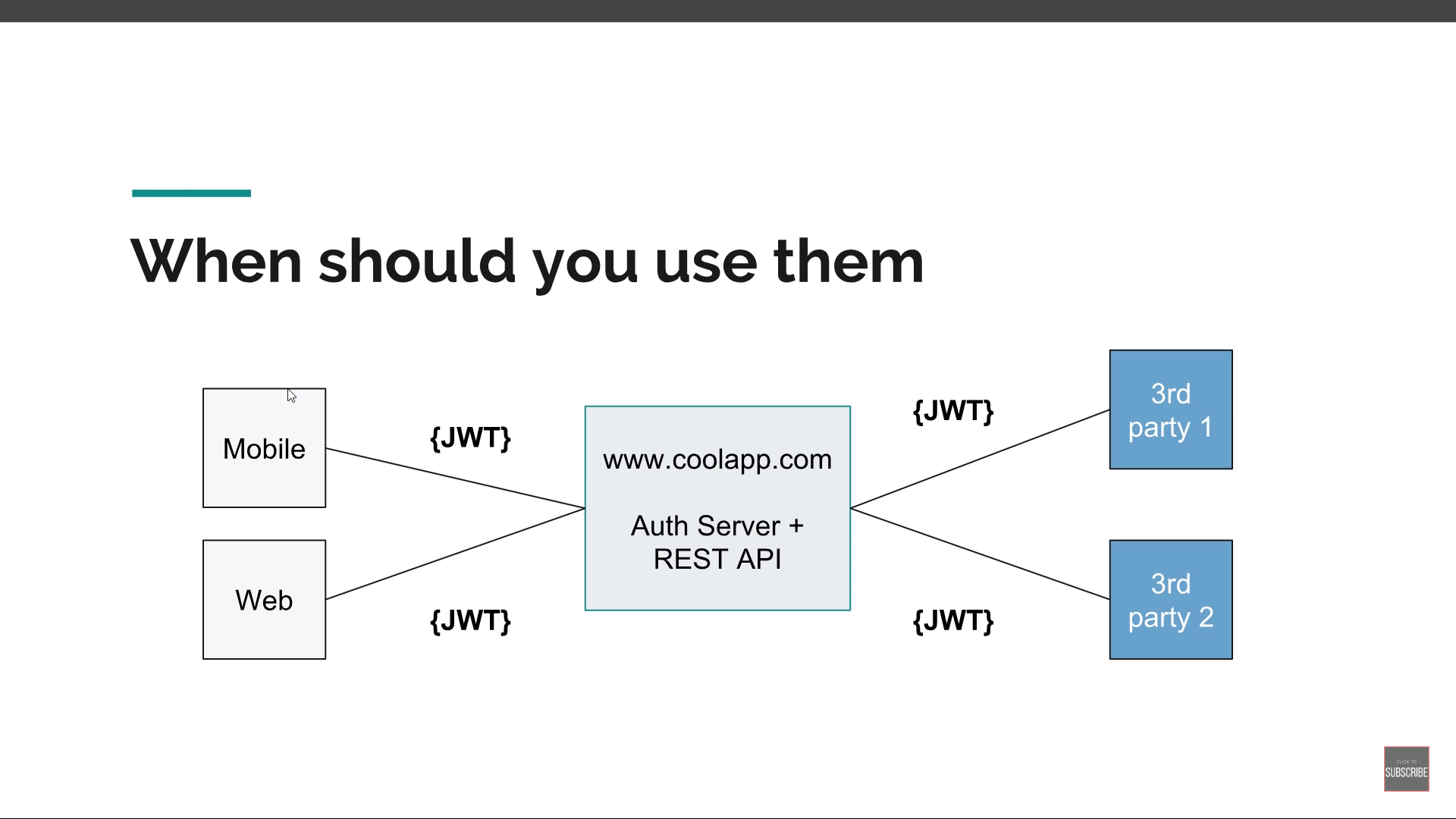
Header

Payload

Secret



When to use: when application is not self contained(microservices)



**SSL AND HTTPS**

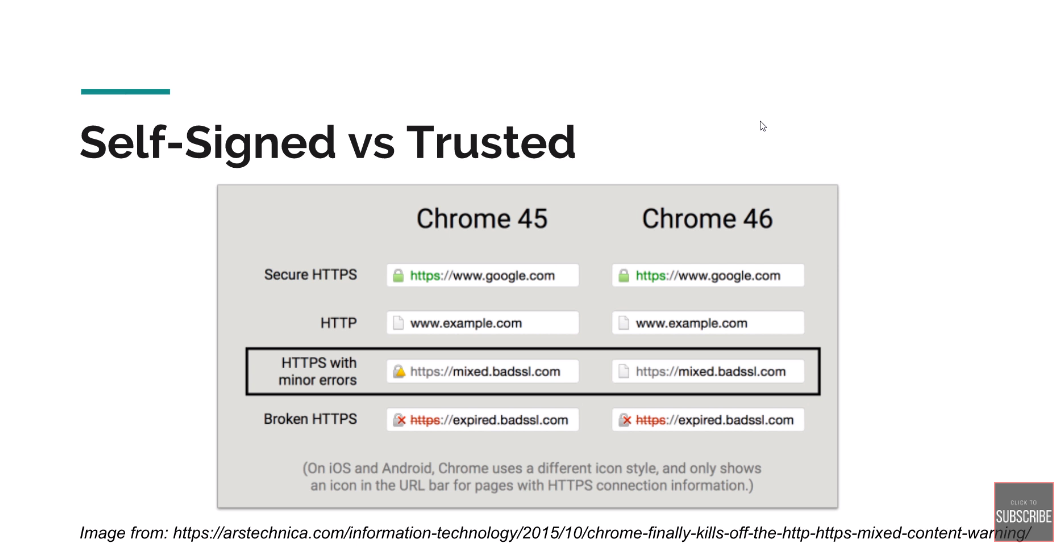
*HTTP+SECURITY LAYER*

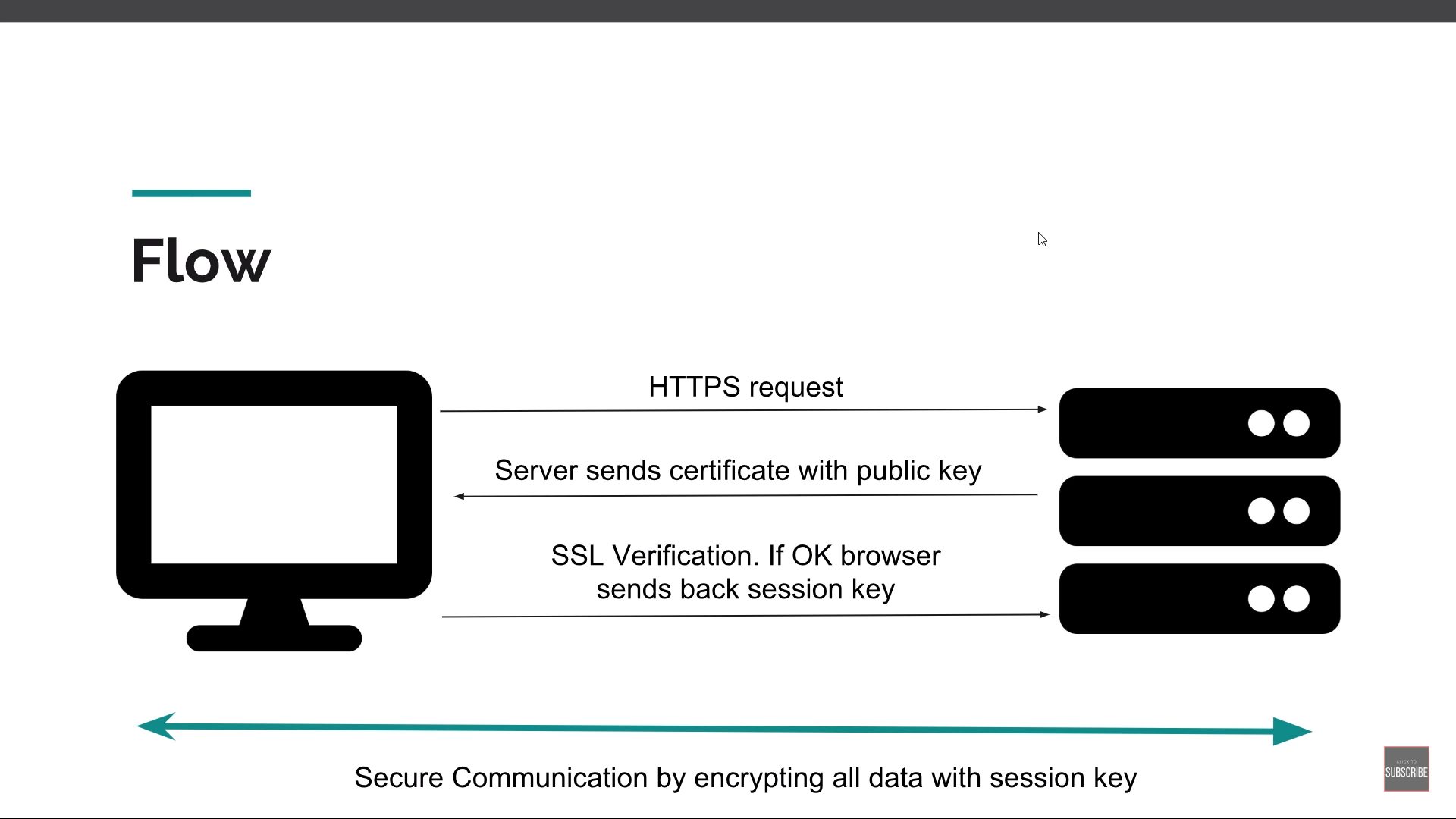
*HTTP IS NOT SECURE*

*With https data transmitted is transmitted in encoded way.*

***SSL Certificates****: It encrypts the data.*

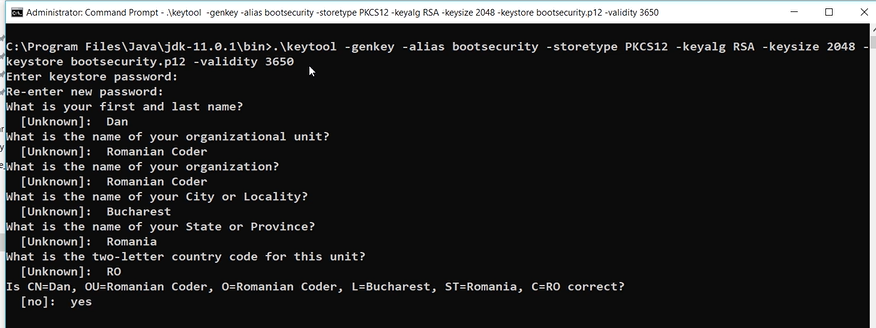
1. *Self-Signed(created by us): good for dev*
2. *Signed by trusted authority: good for prod*





*SSL is really mandatory for any kind of authentication.*

**1.Generate the certificate**



**2.APP.PROPERTIES**

**server.port**=**8443  
server.ssl.enabled**=**true  
server.ssl.key-store**=**src/main/resources/bootsecurity.p12  
server.ssl.key-store-password**=**@kundan621Kk  
server.ssl.key-store-type**=**PKCS12  
server.ssl.key-alias**=**bootsecurity**

**3.in main application**

@Bean  
 **public** ServletWebServerFactory servletContainer() {  
 *// Enable SSL Trafic* TomcatServletWebServerFactory tomcat = **new** TomcatServletWebServerFactory() {  
 @Override  
 **protected void** postProcessContext(Context context) {  
 SecurityConstraint securityConstraint = **new** SecurityConstraint();  
 securityConstraint.setUserConstraint(**"CONFIDENTIAL"**);  
 SecurityCollection collection = **new** SecurityCollection();  
 collection.addPattern(**"/\*"**);  
 securityConstraint.addCollection(collection);  
 context.addConstraint(securityConstraint);  
 }  
 };  
  
 *// Add HTTP to HTTPS redirect* tomcat.addAdditionalTomcatConnectors(httpToHttpsRedirectConnector());  
  
 **return** tomcat;  
 }  
  
 */\*  
 We need to redirect from HTTP to HTTPS. Without SSL, this application used  
 port 8082. With SSL it will use port 8443. So, any request for 8082 needs to be  
 redirected to HTTPS on 8443.  
 \*/* **private** Connector httpToHttpsRedirectConnector() {  
 Connector connector = **new** Connector(TomcatServletWebServerFactory.***DEFAULT\_PROTOCOL***);  
 connector.setScheme(**"http"**);  
 connector.setPort(8082);  
 connector.setSecure(**false**);  
 connector.setRedirectPort(8443);  
 **return** connector;  
 }  
  
}