Architecture of Enterprise Applications 10 Security

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents and Objectives

- Contents
  - Spring Security Samples
  - SECURITY
    - DIGITAL SIGNATURES
    - CODE SIGNING
    - ENCRYPTION
  - Single Sign-On
    - Overview
    - Kerberos protocol
  - Design Tactics of Security

- Objectives
  - 能够根据业务需求，配置使用合理的加密通信方式，并能够理解其基本原理与工作方式

- Spring Security
  - provides comprehensive support for authentication, authorization, and protection against common exploits. It also provides integration with other libraries to simplify its usage.
  - https://docs.spring.io/spring-security/site/docs/current/reference/html5/#features

Spring Security - Login Sample

- MvcConfig.java

```
@Configuration
public class MvcConfig implements WebMvcConfigurer {

  public void addViewControllers(ViewControllerRegistry registry) {
    registry.addViewController("/home").setViewName("home");
    registry.addViewController("/").setViewName("home");
    registry.addViewController("/hello").setViewName("hello");
    registry.addViewController("/login").setViewName("login");
  }

}
```

Login Sample

- WebSecurityConfigureAdapter.java

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http
      .authorizeRequests()
        .antMatchers("/", "/home").permitAll()
        .anyRequest().authenticated()
        .and()
      .formLogin()
        .loginPage("/login")
        .permitAll()
        .and()
      .logout()
        .permitAll();
  }
```

Spring Security - Login Sample

- WebSecurityConfigureAdapter.java

```
@Bean
@Override
public UserDetailsService userDetailsService() {
  UserDetails user =
     User.withDefaultPasswordEncoder()
      .username("user")
      .password("password")
      .roles("USER")
      .build();

  return new InMemoryUserDetailsManager(user);
 }
}
```

Spring Security - Login Sample

REliable, INtelligent & Scalable Systems

- home.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
  <head>
    <title>Spring Security Example</title>
  </head>
  <body>
    <h1>Welcome!</h1>

    <p>Click <a th:href="@{/hello}">here</a> to see a greeting.</p>
  </body>
</html>
```

Login Sample

- hello.html

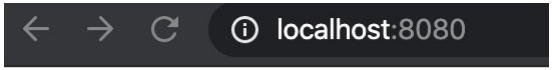
```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="https://www.thymeleaf.org"
    xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1 th:inline="text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>
    <form th:action="@{/logout}" method="post">
      <input type="submit" value="Sign Out"/>
    </form>
  </body>
</html>
```

Login Sample

- login.html

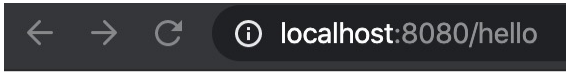
```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
    xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
  <head>
    <title>Spring Security Example </title>
  </head>
  <body>
    <div th:if="${param.error}">
      Invalid username and password.
    </div>
    <div th:if="${param.logout}">
      You have been logged out.
    </div>
    <form th:action="@{/login}" method="post">
      <div><label> User Name : <input type="text" name="username"/> </label></div>
      <div><label> Password: <input type="password" name="password"/> </label></div>
      <div><input type="submit" value="Sign In"/></div>
    </form>
  </body>
</html>
```

Login Sample



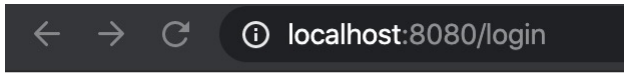
Welcome!

Click here to see a greeting.

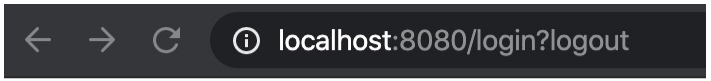


Hello user!

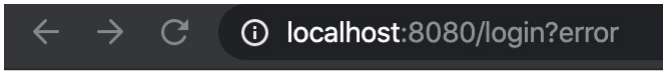
Sign Out



User Name :
Password:
Sign In



You have been logged out.
User Name :
Password:
Sign In



Invalid username and password.
User Name :
Password:
Sign In

React + Spring Security: Front-end

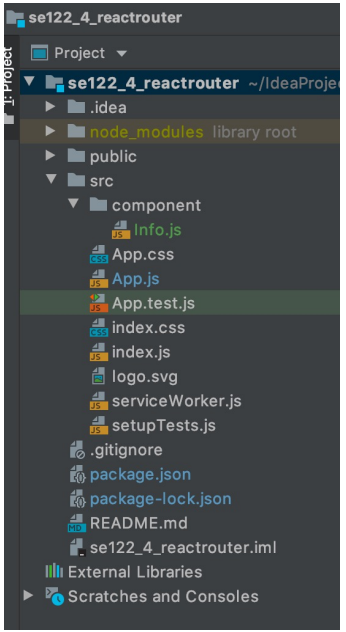
- App.js

```
import React from 'react';
import {BrowserRouter as Router, Switch, Route, Link} from "react-router-dom";
import Info from './component/Info';
function App() {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li><Link to="/">Home</Link></li>
            <li><Link to="/about">About</Link></li>
            <li><Link to="/users">Users</Link></li>
          </ul>
        </nav>
        <Switch>
          <Route path="/about"><Info menu="about"/></Route>
          <Route path="/users"><Info menu="users"/></Route>
          <Route path="/"><Info menu=""/></Route>
        </Switch>
      </div>
    </Router>
  );
}
export default App;
```

React + Spring Security: Front-end

- Info.js

```
import React from 'react';
function Info(props) {
  let url = 'http://localhost:8080/' + props.menu;
  let username = 'root';
  let password = '123';
  let headers = new Headers();
  headers.set('Authorization', 'Basic ' + Buffer.from(username + ":" + password).toString('base64'));

  fetch(url, {
    method: 'GET',
    headers: headers,
    credentials: 'include'
  }).then(response => response.text())
    .then(data => {
      document.getElementById("info").innerText = data
    }).catch(function (ex) {
    console.log('parsing failed', ex)
  })
  return (
    <div>
      <h1 id="info">Welcome</h1>
    </div>
  );
}
export default Info;
```



React + Spring Security: Back-end

- SpringSecurityApplication.java

```
@SpringBootApplication(exclude= {DataSourceAutoConfiguration.class})
public class SpringSecurityApplication {
   public static void main(String[] args) {
      SpringApplication.run(SpringSecurityApplication.class, args);
   }
}
```

- application.properties

```
spring.security.user.name=root
spring.security.user.password=123
```

React + Spring Security: Back-end

- GreetingController.java

```
@CrossOrigin(maxAge = 3600)
@RestController
public class GreetingController {
  @GetMapping("/about")
  public String getAbout() {
    return "This is a Spring security sample";
  }

  @GetMapping("/users")
  public String getUser() {
    return "I am a user";
  }

  @GetMapping("/")
  public String getHome() {
    return "Let' start!";
  }
}
```

React + Spring Security: Back-end

REliable, INtelligent & Scalable Systems

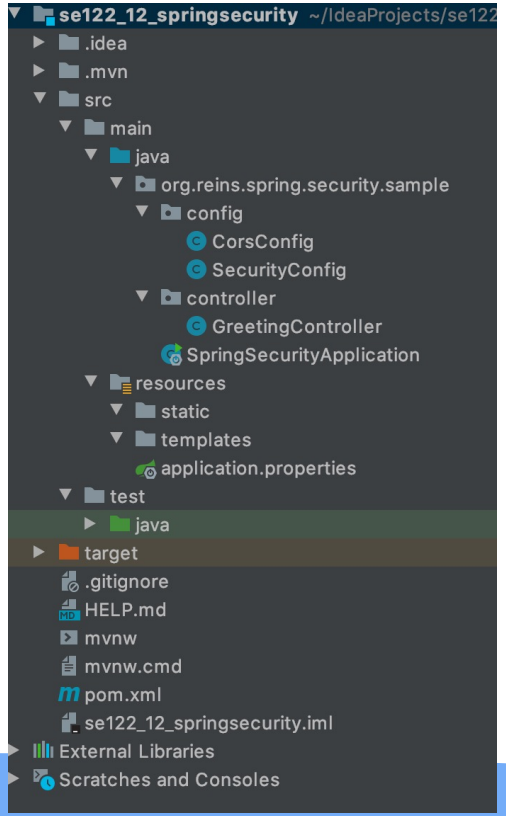
- SecurityConfig.java

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http.cors()
            .and()
            .authorizeRequests(authorize -> authorize
                  .antMatchers("/").permitAll()
                  .antMatchers("/users","/about").authenticated()
             )
             .httpBasic(withDefaults())
   }
   @Bean
   public CorsConfigurationSource corsConfigurationSource() {
      final CorsConfiguration configuration = new CorsConfiguration();
      configuration.setAllowedOrigins(ImmutableList.of("*"));
      configuration.setAllowedMethods(ImmutableList.of("HEAD", "GET", "POST", "PUT", "DELETE", "PATCH"));
      configuration.setAllowCredentials(true);
      configuration.setAllowedHeaders(ImmutableList.of("Authorization", "Cache-Control", "Content-Type"));
      final UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
      source.registerCorsConfiguration("/**", configuration);
      return source;
   }
}
```

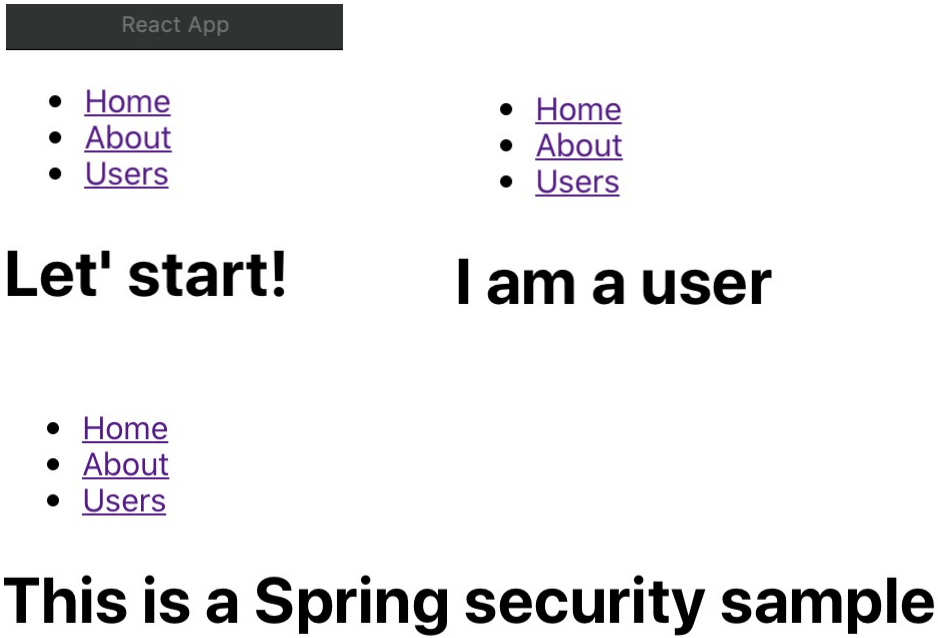
React + Spring Security: Back-end

- CorsConfig.java

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("*")
                .allowedMethods("*")
                .allowedHeaders("*")
                .exposedHeaders(HttpHeaders.SET_COOKIE)
                .allowCredentials(true).maxAge(1800);
    }
}
```



Run the application



Digital Signatures

- To give more trust to an applet, we need to know two things:
  - Where did the applet come from?
  - Was the code corrupted in transit?

Message Digests

- A message digest is a digital fingerprint of a block of data.
  - For example, the so-called SHA1 (secure hash algorithm #1) condenses any data block, no matter how long, into a sequence of 160 bits (20 bytes).

- A message digest has two essential properties:
  - If one bit or several bits of the data are changed, then the message digest also changes.
  - A forger who is in possession of a given message cannot construct a fake message that has the same message digest as the original.

Message Digests

- Consider the following message by the billionaire father:
  - "Upon my death, my property shall be divided equally among my children; however, my son George shall receive nothing."
  - That message has an SHA1 fingerprint of
    - 2D 8B 35 F3 BF 49 CD B1 94 04 E0 66 21 2B 5E 57 70 49 E1 7E
  - Now, suppose George wants to change the message so that Bill gets nothing. That changes the fingerprint to a completely different bit pattern:
    - 2A 33 0B 4B B3 FE CC 1C 9D 5C 01 A7 09 51 0B 49 AC 8F 98 92

Message Digests


```
MessageDigest alg = MessageDigest.getInstance("SHA-1");

InputStream in = . . .
int ch;
while ((ch = in.read( )) != -1)
   alg.update((byte) ch);

byte[ ] bytes = . . .;
alg.update(bytes);

byte[ ] hash = alg.digest( );
```

Message Signing

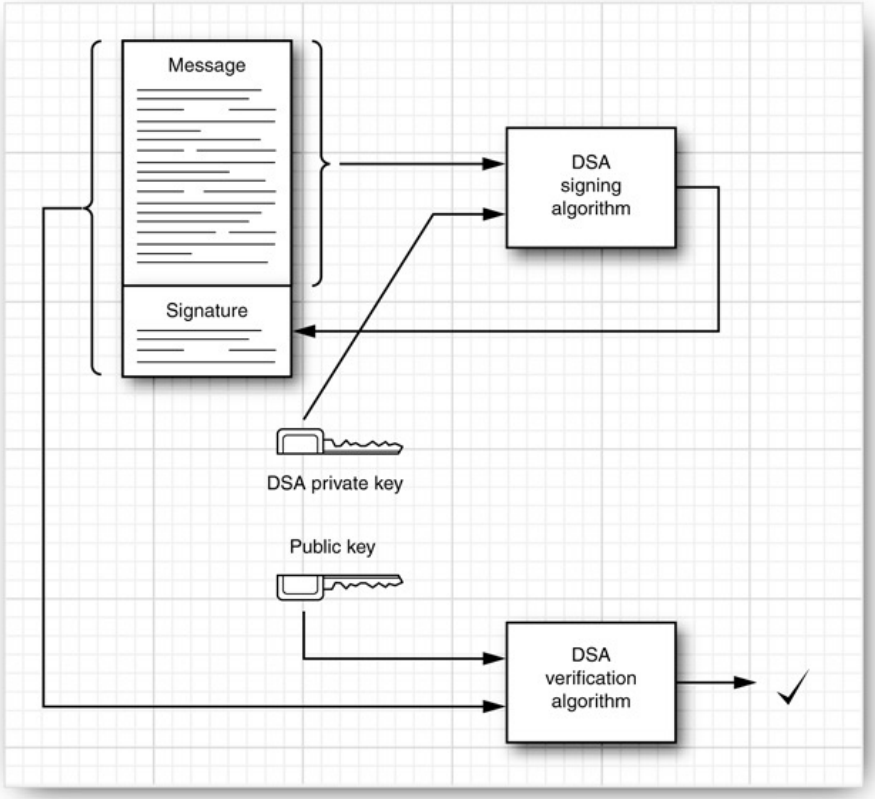
- The message digest algorithms are publicly known, and they don't require secret keys.
  - In that case, the recipient of the forged message and the recomputed fingerprint would never know that the message has been altered.
  - Digital signatures solve this problem.

Message Signing

- The keys are quite long and complex. For example, here is a matching pair of public and private Digital Signature Algorithm (DSA) keys.
- Public key:
- Code View:
  - p: fca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12ed0899 bcd132acd50d99151bdc43ee737592e17 q: 962eddcc369cba8ebb260ee6b6a126d9346e38c5 g:678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e29356 30e 1c2062354d0da20a6c416e50be794ca4 y: c0b6e67b4ac098eb1a32c5f8c4c1f0e7e6fb9d832532e27d0bdab9ca2d2a8123ce5a8018b8161a760480fadd040b927 281ddb22cb9bc4df596d7de4d1b977d50

- Private key:
- Code View:
  - p: fca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12ed0899 bcd132acd50d99151bdc43ee737592e17 q: 962eddcc369cba8ebb260ee6b6a126d9346e38c5 g: 678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e2935630 e1c2062354d0da20a6c416e50be794ca4 x: 146c09f881656cc6c51f27ea6c3a91b85ed1d70a

Message Signing



X.509 Certificate

- To take advantage of public key cryptography, the public keys must be distributed.
  - One of the most common distribution formats is called X.509.

- The keytool program manages keystores, databases of certificates and private/public key pairs.
  - Each entry in the keystore has an alias.
  - Here is how Alice creates a keystore, alice.certs, and generates a key pair with alias alice.
  - keytool -genkeypair -keystore alice.certs -alias alice

X.509 Certificate

- When generating a key, you are prompted for the following information:

```
Enter keystore password:  password
What is your first and last name?
  [Unknown]:  Alice Lee
What is the name of your organizational unit?
  [Unknown]:  Engineering Department
What is the name of your organization?
  [Unknown]:  ACME Software
What is the name of your City or Locality?
  [Unknown]:  Cupertino
What is the name of your State or Province?
  [Unknown]:  California
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=Alice Lee, OU=Engineering Department, O=ACME Software, L=Cupertino, ST=California,
C=US> correct?
[no]:  Y
```

X.509 Certificate

- Alice exports a certificate file:
  - keytool -exportcert -keystore alice.certs -alias alice -file alice.cer
- Bob receives the certificate, he can print it:
  - keytool -printcert -file alice.cer
- The printout looks like this:

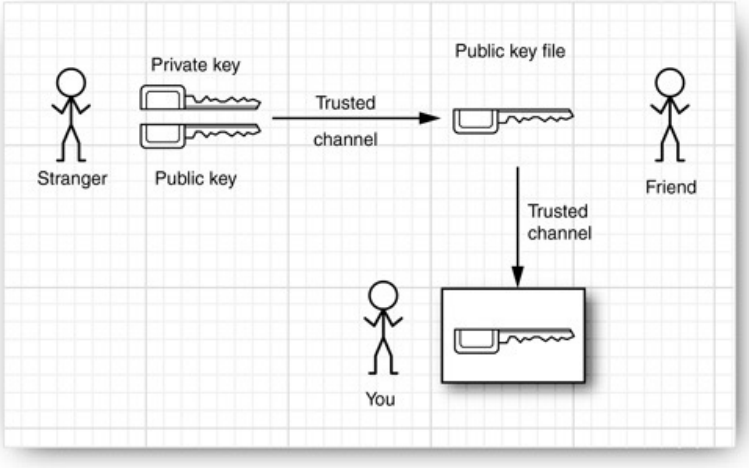
Owner: CN=Alice Lee, OU=Engineering Department, O=ACME Software,
L=San Francisco, ST=CA, C=US
Issuer: CN=Alice Lee, OU=Engineering Department, O=ACME Software,
L=San Francisco, ST=CA, C=US
Serial number: 470835ce
Valid from: Sat Oct 06 18:26:38 PDT 2007 until: Fri Jan 04 17:26:38 PST 2008
Certificate fingerprints:
MD5: BC:18:15:27:85:69:48:B1:5A:C3:0B:1C:C6:11:B7:81
SHA1: 31:0A:A0:B8:C2:8B:3B:B6:85:7C:EF:C0:57:E5:94:95:61:47:6D:34
Signature algorithm name: SHA1withDSA
Version: 3

X.509 Certificate

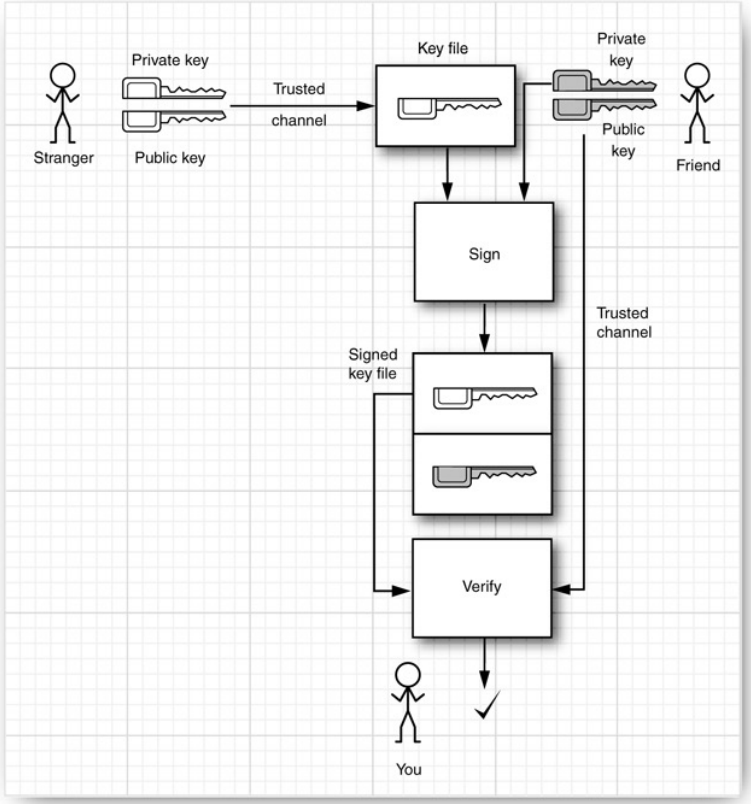
- Once Bob trusts the certificate, he can import it into his keystore.
  - keytool -importcert -keystore bob.certs -alias alice -file alice.cer
- Now Alice can start sending signed documents to Bob.
  - jar cvf document.jar document.txt
  - jarsigner -keystore alice.certs document.jar alice
- When Bob receives the file, he uses the -verify option of the jarsigner program.
  - jarsigner -verify -keystore bob.certs document.jar
- If the JAR file is not corrupted and the signature matches, then the jarsigner program prints
  - jar verified.
  - Otherwise, the program displays an error message.

Authentication Problem

- Be careful:
  - You still have no idea who wrote the message. Anyone could have generated a pair of public and private keys, signed the message with the private key, and sent the signed message and the public key to you.
  - The problem of determining the identity of the sender is called the authentication problem.



Authentication Problem



Certificate Signing

- Suppose Alice wants to send her colleague Cindy a signed message
  - but Cindy doesn't want to bother with verifying lots of signature fingerprints.
  - Now suppose that there is an entity that Cindy trusts to verify signatures. In this example, Cindy trusts the Information Resources Department at ACME Software.
- That department operates a certificate authority (CA).
  - Everyone at ACME has the CA's public key in their keystore, installed by a system administrator who carefully checked the key fingerprint.
  - The CA signs the keys of ACME employees.
  - When they install each other's keys, then the keystore will trust them implicitly because they are signed by a trusted key.

Certificate Signing

- Here is how you can simulate this process.
  - Create a keystore acmesoft.certs.
  - Generate a key par and export the public key:
    - keytool -genkeypair -keystore acmesoft.certs -alias acmeroot
    - keytool -exportcert -keystore acmesoft.certs -alias acmeroot -file acmeroot.cer
  - The public key is exported into a "self-signed" certificate.
  - Then add it to every employee's keystore.
    - keytool -importcert -keystore cindy.certs -alias acmeroot -file acmeroot.cer
  - An authorized staff member at ACME Software would verify Alice's identity and generate a signed certificate as follows:
    - java CertificateSigner -keystore acmesoft.certs -alias acmeroot -infile alice.cer -outfile alice_signedby_acmeroot.cer
  - Now Cindy imports the signed certificate into her keystore:
    - keytool -importcert -keystore cindy.certs -alias alice -file alice_signedby_acmeroot.cer

Code Signing

- One of the most important uses of authentication technology is signing executable programs.

- You now know how to implement this sophisticated scheme.
  - Use authentication to verify where the code came from.
  - Run the code with a security policy that enforces the permissions that you want to grant the program, depending on its origin.

JAR File Signing

- ACME decides to sign the JAR files that contain the program code.
  - First, ACME generates a root certificate:
    - keytool -genkeypair -keystore acmesoft.certs -alias acmeroot
  - Therefore, we create a second keystore client.certs for the public certificates and add the public acmeroot certificate into it.
    - keytool -exportcert -keystore acmesoft.certs -alias acmeroot -file acmeroot.cer
    - keytool -importcert -keystore client.certs -alias acmeroot -file acmeroot.cer
  - To make a signed JAR file, programmers add their class files to a JAR file in the usual way. For example,
    - javac FileReadApplet.java
    - jar cvf FileReadApplet.jar *.class
  - Then a trusted person at ACME runs the jarsigner tool, specifying the JAR file and the alias of the private key:
    - jarsigner -keystore acmesoft.certs FileReadApplet.jar acmeroot

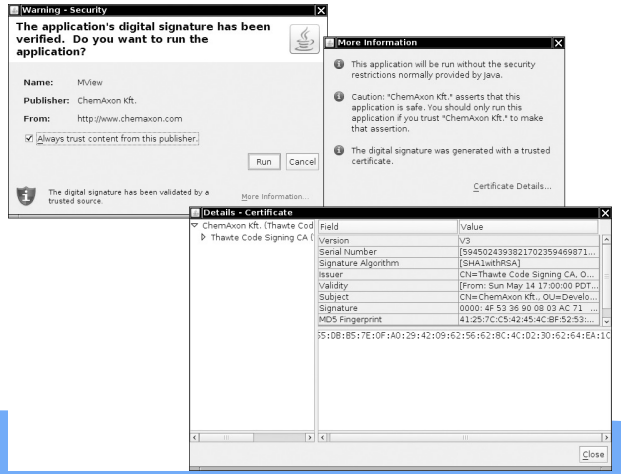
JAR File Signing

- ACME decides to sign the JAR files.
  - Next, let us turn to the client machine configuration. A policy file must be distributed to each client machine.
  - To reference a keystore, a policy file starts with the line
    - keystore "keystoreURL", "keystoreType";
  - The URL can be absolute or relative.
    - keystore "client.certs", "JKS";
  - Then grant clauses can have suffixes signedBy "alias", such as this one:
    - grant signedBy "acmeroot" { . . . };
  - Now create a policy file applet.policy with the contents:
    - keystore "client.certs", "JKS";
    - grant signedBy "acmeroot" {
    - permission java.lang.RuntimePermission "usePolicy";
    - permission java.io.FilePermission "/etc/*", "read";
    - };

Software Developer Certificates

REliable, INtelligent & Scalable Systems

- A program signed with a software developer certificate that is issued by a CA will trigger a pop-up dialog box identifies the software developer and the certificate issuer.
  - You now have two choices:
    - Run the program with full privileges.
    - Confine the program to the sandbox. (The Cancel button in the dialog box is misleading. If you click that button, the applet is not canceled. Instead, it runs in the sandbox.)



Symmetric Ciphers

- Cipher

  Cipher cipher = Cipher.getInstance(algorithmName);

  – or

  Cipher cipher = Cipher.getInstance(algorithmName, providerName);

  – The JDK comes with ciphers by the provider named "SunJCE".

  – The algorithm name is a string such as "AES" or "DES/CBC/PKCS5Padding".

  int mode = . . .; Key key = . . .; cipher.init(mode, key);

  – The mode is one of

  Cipher.ENCRYPT_MODE

  Cipher.DECRYPT_MODE

  Cipher.WRAP_MODE

  Cipher.UNWRAP_MODE

Symmetric Ciphers

```
int blockSize = cipher.getBlockSize();
byte[] inBytes = new byte[blockSize];
. . . // read inBytes
int outputSize= cipher.getOutputSize(inLength);
byte[] outBytes = new byte[outputSize];
int outLength = cipher.update(inBytes, 0, outputSize, outBytes);
. . . // write outBytes

outBytes = cipher.doFinal(inBytes, 0, inLength);
```

- Or

```
outBytes = cipher.doFinal();
```

- The call to doFinal is necessary to carry out padding of the final block.

```
L 01                        if length(L) = 7
L 02 02                     if length(L) = 6
L 03 03 03                  if length(L) = 5
. . .
L 07 07 07 07 07 07 07  if length(L) = 1
08 08 08 08 08 08 08 08
```

Key generation

- Follow these steps:
  - Get a KeyGenerator for your algorithm.
  - Initialize the generator with a source for randomness. If the block length of the cipher is variable, also specify the desired block length.
  - Call the generateKey method.

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
SecureRandom random = new SecureRandom();
keygen.init(random);
Key key = keygen.generateKey();
```

Or

```
SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("AES");
byte[] keyData = . . .; // 16 bytes for AES
SecretKeySpec keySpec = new SecretKeySpec(keyData, "AES");
Key key = keyFactory.generateSecret(keySpec);
```

Cipher Streams

- The JCE library provides a convenient set of stream classes that automatically encrypt or decrypt stream data.
- Encryption

```
Cipher cipher = . . .;
cipher.init(Cipher.ENCRYPT_MODE, key);
CipherOutputStream out = new CipherOutputStream(new FileOutputStream(outputFileName),
cipher);
byte[] bytes = new byte[BLOCKSIZE];
int inLength = getData(bytes); // get data from data source
while (inLength != -1) {
    out.write(bytes, 0, inLength);
    inLength = getData(bytes); // get more data from data source
} out.flush();
```

Cipher Streams

- The JCE library provides a convenient set of stream classes that automatically encrypt or decrypt stream data.
- Decryption

```
Cipher cipher = . . .;
cipher.init(Cipher.DECRYPT_MODE, key);
CipherInputStream in = new CipherInputStream(new FileInputStream(inputFileName), cipher);
byte[] bytes = new byte[BLOCKSIZE];
int inLength = in.read(bytes);
while (inLength != -1) {
    putData(bytes, inLength); // put data to destination
    inLength = in.read(bytes);
}
```

Public Key Ciphers

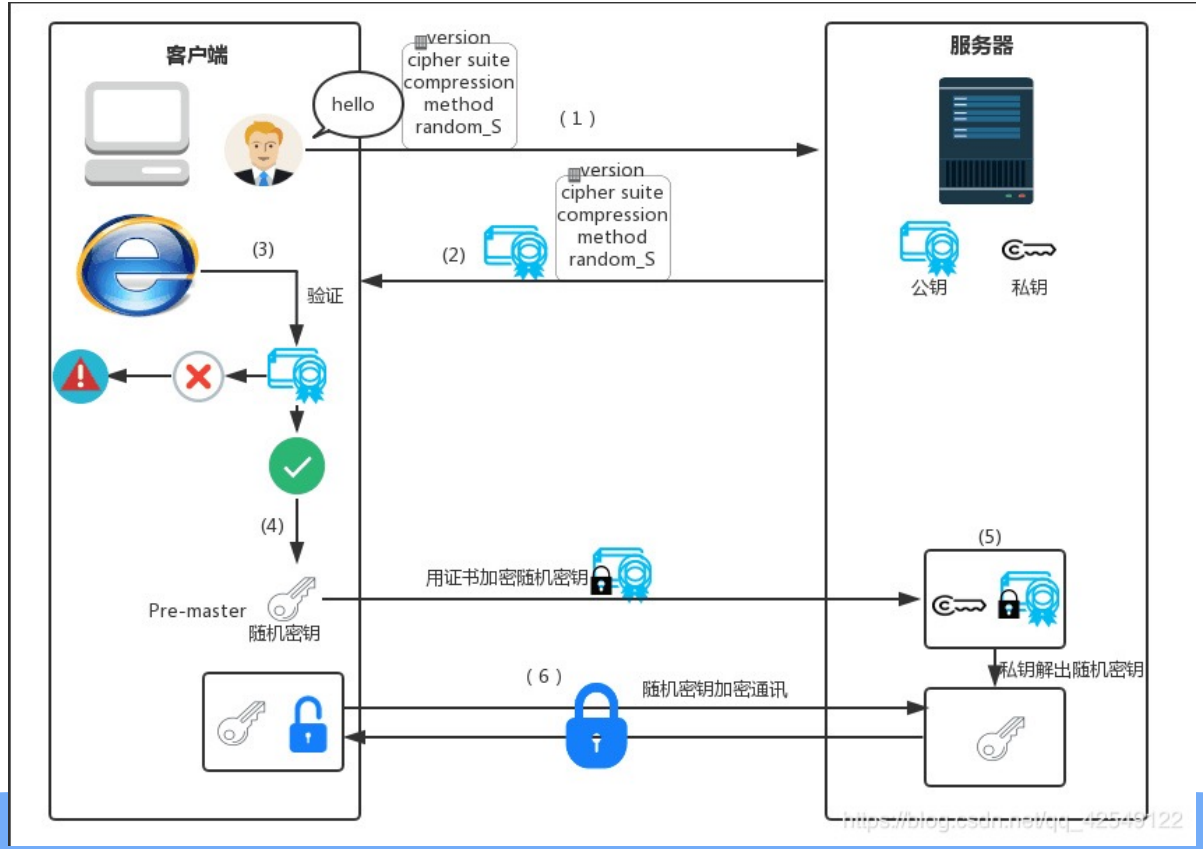
- The Achilles heel of symmetric ciphers is key distribution.
  - Public key cryptography solves that problem.
- All known public key algorithms are much slower than symmetric key algorithms such as DES or AES.
  - It would not be practical to use a public key algorithm to encrypt large amounts of information.
- This problem can easily be overcome by combining a public key cipher with a fast symmetric cipher, like this:
  - Alice generates a random symmetric encryption key. She uses it to encrypt her plaintext.
  - Alice encrypts the symmetric key with Bob's public key.
  - Alice sends Bob both the encrypted symmetric key and the encrypted plaintext.
  - Bob uses his private key to decrypt the symmetric key.
  - Bob uses the decrypted symmetric key to decrypt the message.

Introduction to SSL/TLS

- Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL),
  - are technologies which allow web browsers and web servers to communicate over a secured connection.
  - This means that the data being sent is encrypted by one side, transmitted, then decrypted by the other side before processing.
  - This is a two-way process, meaning that both the server AND the browser encrypt all traffic before sending out data.
- Another important aspect of the SSL/TLS protocol is Authentication.
  - This means that during your initial attempt to communicate with a web server over a secure connection, that server will present your web browser with a set of credentials, in the form of a "Certificate", as proof the site is who and what it claims to be.
  - In certain cases, the server may also request a Certificate from your web browser, asking for proof that *you* are who you claim to be.
  - This is known as "Client Authentication", although in practice this is used more for business-to-business (B2B) transactions than with individual users. Most SSL-enabled web servers do not request Client Authentication.

SSL in Tomcat

- https://blog.csdn.net/qq_42549122/article/details/90272299



Configuration in Tomcat

- Create a keystore file to store the server's private key and self-signed certificate by executing the following command:
  - Windows:
  - "%JAVA_HOME%\bin\keytool" -genkey -alias tomcat -keyalg RSA -keystore "C:\Tomcat\conf\key\tomcat.keystore" -validity 365
  - Unix:
  - $JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA -keystore ./conf/key/tomcat.keystore -validity 365

Configuration in Tomcat

REliable, INtelligent & Scalable Systems

- For External Tomcat -> Edit the Tomcat Configuration File ./conf/server.xml

```
<Connector port="8443"
 protocol="org.apache.coyote.http11.Http11NioProtocol"
 maxThreads="150" SSLEnabled="true"
 keystoreFile="/Users/chenhaopeng/apache-tomcat-9.0.31/conf/key/tomcat.keystore"
 keystorePass="changeit">
</Connector>
```

- For Spring nested Tomcat -> Edit application.properties

```
server.port=8443
server.ssl.key-store=/Users/chenhaopeng/apache-tomcat-9.0.31/conf/key/tomcat.keystore
server.ssl.key-store-password=changeit
server.ssl.keyAlias=tomcat
```

Test SSL in Tomcat

- Spring-boot Project

```
@SpringBootApplication
public class DemoApplication {

   @Bean
   public TomcatServletWebServerFactory tomcatServletWebServerFactory(Connector connector){
      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);
         }
      };
      tomcat.addAdditionalTomcatConnectors(connector);
      return tomcat;
   }
```

Test SSL in Tomcat

- Spring-boot Project

```
@SpringBootApplication
public class DemoApplication {

    @Bean
    public Connector connector(){
        Connector connector=new Connector("org.apache.coyote.http11.Http11NioProtocol");
        connector.setScheme("http");
        connector.setPort(8080);
        connector.setSecure(false);
        connector.setRedirectPort(8443);
        return connector;
    }

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}
```

Test SSL in Tomcat

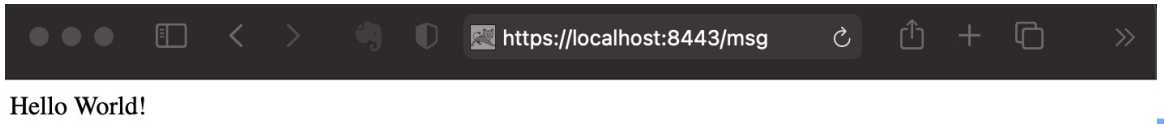
- MsgController.java

```
@RestController
public class MsgController {
   @Autowired
   WebApplicationContext applicationContext;

   @GetMapping(value = "/msg")
   public String findOne( ) {
      System.out.println("Sending an email message.");
      return "Hello World!";
   };
}
```

Test SSL in Tomcat

- MsgController.java

```
@RestController
public class MsgController {
   @Autowired
   WebApplicationContext applicationContext;

   @GetMapping(value = "/msg")
   public String findOne( ) {
      System.out.println("Sending an email message.");
      return "Hello World!";
   };
}
```



Single Sign-On - wikipedia

- **Single sign-on** (**SSO**) is a property of access control of multiple related, but independent software systems.
  - With this property a user logs in once and gains access to all systems without being prompted to log in again at each of them.
  - Conversely, **Single sign-off** is the property whereby a single action of signing out terminates access to multiple software systems.

- As different applications and resources support different authentication mechanisms,
  - single sign-on has to internally translate to and store different credentials compared to what is used for initial authentication.

Single Sign-On - wikipedia

- **Benefits** include:
  - Reduces phishing success, because users are not trained to enter password everywhere without thinking.
  - Reducing password fatigue from different user name and password combinations
  - Reducing time spent re-entering passwords for the same identity
  - Reducing IT costs due to lower number of IT help desk calls about passwords
  - Security on all levels of entry/exit/access to systems without the inconvenience of re-prompting users
  - Centralized reporting for compliance adherence.

Common Single Sign-on

- Kerberos based
  - MIT Kerberos protocol
- Smart card based
  - Initial sign-on prompts the user for the smart card.
  - Additional software applications also use the smart card, without prompting the user to re-enter credentials.
  - Smart card-based single sign-on can either use certificates or passwords stored on the smart card.
- OTP token
  - Also referred to as one-time password token.
- Security Assertion Markup Language
  - Security Assertion Markup Language (SAML) is an XML-based solution for exchanging user security information between an enterprise and a service provider.

Kerberos

- Kerberos
  - is a computer network authentication protocol which works on the basis of "tickets" to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner.

- MIT developed Kerberos to protect network services provided by Project Athena.
  - The protocol was named after the character *Kerberos* (or *Cerberus*) from Greek mythology which was a monstrous three-headed guard dog of Hades.

- **User Client-based Logon**
  - A user enters a username and password on the client machines.
  - The client performs a one-way function (hash usually) on the entered password, and this becomes the secret key of the client/user.

- **Client Authentication**
  - The client sends a clear text message of the user ID to the AS requesting services on behalf of the user. (Note: Neither the secret key nor the password is sent to the AS.)
  - The AS generates the secret key by hashing the password of the user found at the database (e.g. Active Directory in Windows Server).

- The AS checks to see if the client is in its database. If it is, the AS sends back the following two messages to the client:
  - Message A: *Client/TGS Session Key* encrypted using the secret key of the client/user.
  - Message B: *Ticket-Granting-Ticket* (which includes the client ID, client network address, ticket validity period, and the *client/TGS session key*) encrypted using the secret key of the TGS.

- Once the client receives messages A and B, it attempts to decrypt message A with the secret key generated from the password entered by the user.
  - If the user entered password does not match the password in the AS database, the client's secret key will be different and thus unable to decrypt message A.
  - With a valid password and secret key the client decrypts message A to obtain the *Client/TGS Session Key*. This session key is used for further communications with the TGS. (Note: The client cannot decrypt Message B, as it is encrypted using TGS's secret key.)
  - At this point, the client has enough information to authenticate itself to the TGS.

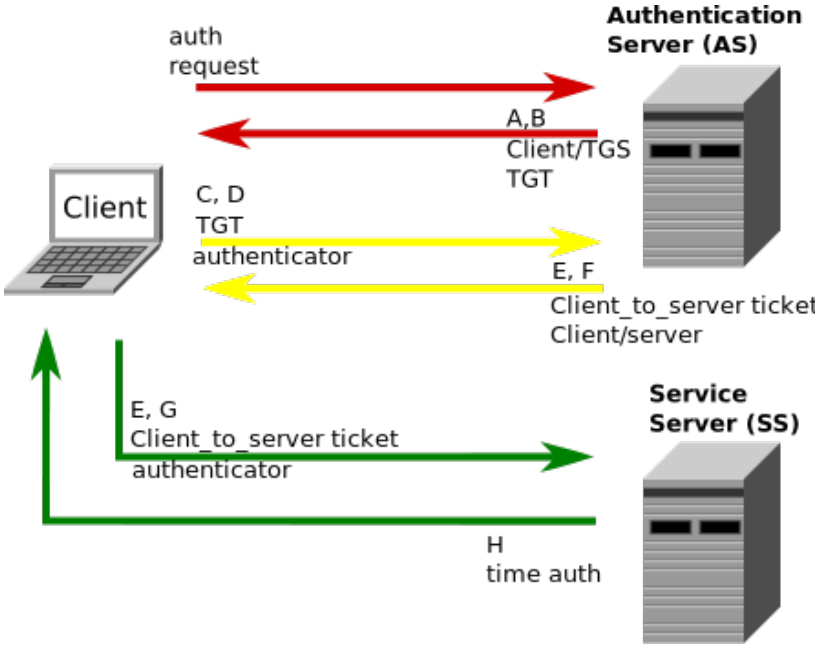
- **Client Service Authorization**
- When requesting services, the client sends the following two messages to the TGS:
  - Message C: Composed of the TGT from message B and the ID of the requested service.
  - Message D: Authenticator (which is composed of the client ID and the timestamp), encrypted using the *Client/TGS Session Key*.

- Upon receiving messages C and D, the TGS retrieves message B out of message C. It decrypts message B using the TGS secret key. This gives it the "client/TGS session key". Using this key, the TGS decrypts message D (Authenticator) and sends the following two messages to the client:
  - Message E: *Client-to-server ticket* (which includes the client ID, client network address, validity period and *Client/Server Session Key*) encrypted using the service's secret key.
  - Message F: *Client/Server Session Key* encrypted with the *Client/TGS Session Key*.

- **Client Service Request**
- Upon receiving messages E and F from TGS, the client has enough information to authenticate itself to the SS. The client connects to the SS and sends the following two messages:
  - Message E from the previous step (the *client-to-server ticket*, encrypted using service's secret key).
  - Message G: a new Authenticator, which includes the client ID, timestamp and is encrypted using *Client/Server Session Key*.

- The SS decrypts the ticket using its own secret key to retrieve the *Client/Server Session Key*. Using the sessions key, SS decrypts the Authenticator and sends the following message to the client to confirm its true identity and willingness to serve the client:
  - Message H: the timestamp found in client's Authenticator plus 1, encrypted using the *Client/Server Session Key*.

- The client decrypts the confirmation using the *Client/Server Session Key* and checks whether the timestamp is correctly updated. If so, then the client can trust the server and can start issuing service requests to the server.
- The server provides the requested services to the client.


auth
request
Authentication
Server (AS)
A,B
Client/TGS
TGT
Client
C, D
TGT
authenticator
E, F
Client_to_server ticket
Client/server
E, G
Client_to_server ticket
authenticator
Service
Server (SS)
H
time auth

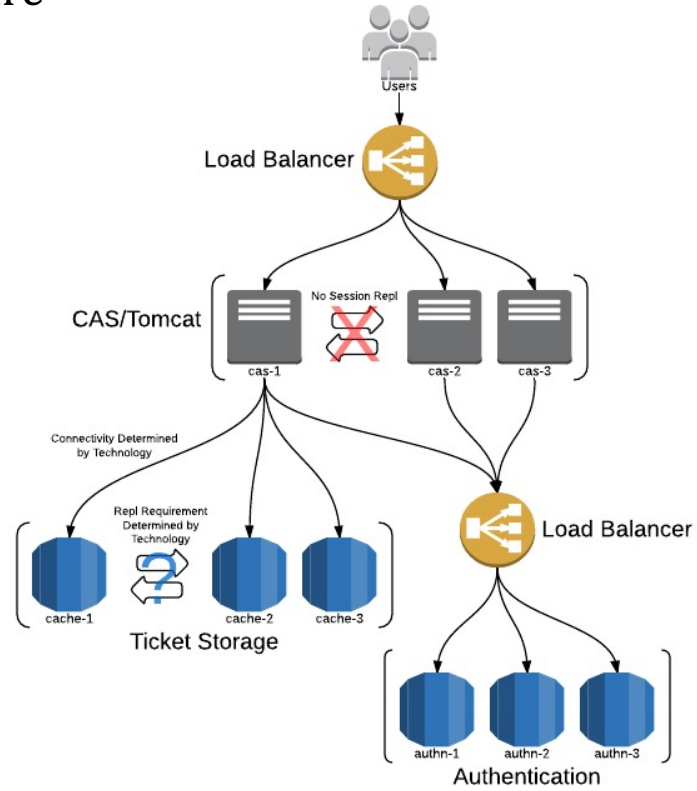
- Drawbacks and Limitations
  - Single point of failure.
  - Kerberos has strict time requirements, which means the clocks of the involved hosts must be synchronized within configured limits.
  - The administration protocol is not standardized and differs between server implementations.
  - Since all authentication is controlled by a centralized KDC, compromise of this authentication infrastructure will allow an attacker to impersonate any user.
  - Each network service which requires a different host name will need its own set of Kerberos keys. This complicates virtual hosting and clusters.

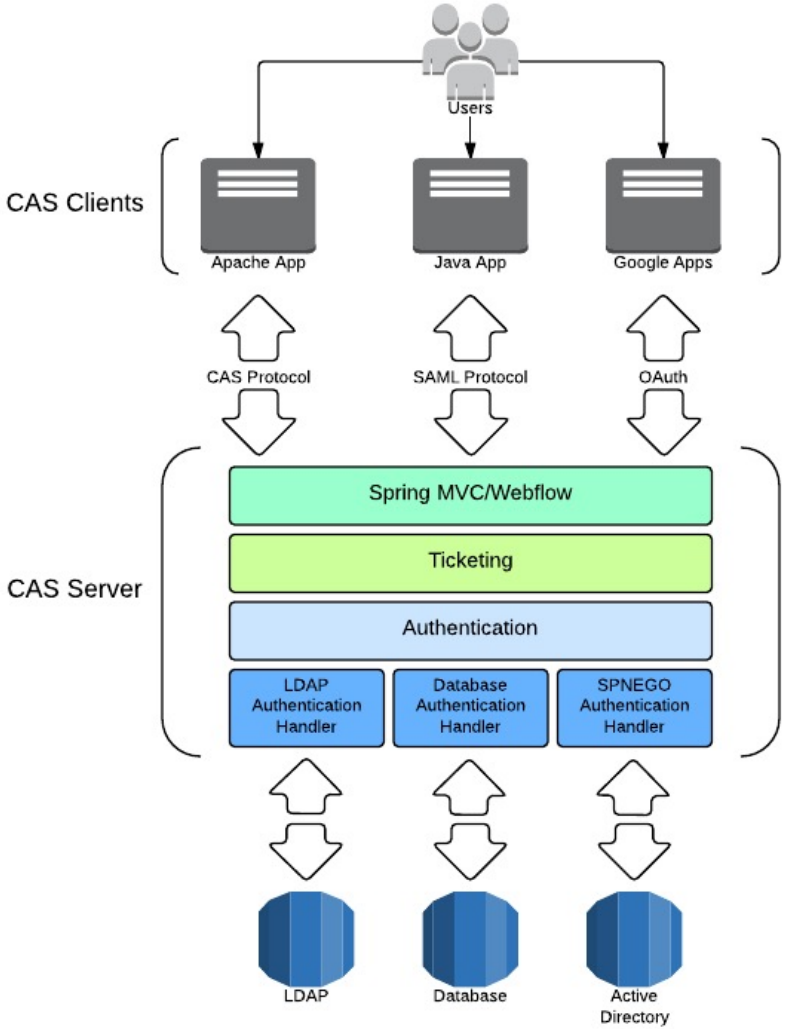
- Enterprise Single Sign-On - CAS provides
  - a friendly open source community that actively supports and contributes to the project.
  - While the project is rooted in higher-ed open source, it has grown to an international audience spanning Fortune 500 companies and small special-purpose installations.

- CAS provides enterprise single sign-on service for the Web:
  - An open and well-documented protocol
  - An open-source Java server component
  - Pluggable authentication support (LDAP, database, X.509, 2-factor)
  - Support for multiple protocols (CAS, SAML, OAuth, OpenID)
  - A library of clients for Java, .Net, PHP, Perl, Apache, uPortal, and others
  - Integrates with uPortal, BlueSocket, TikiWiki, Mule, Liferay, Moodle and others
  - Community documentation and implementation support
  - An extensive community of adopters

CAS

- Architecture





Security Characteristics

- Security can be characterized as
  - a system providing nonrepudiation, confidentiality, integrity, assurance, availability, and auditing.

- Nonrepudiation
  - is the property that a transaction (access to or modification of data or services) cannot be denied by any of the parties to it.
  - This means you cannot deny that you ordered that item over the Internet if, in fact, you did.

- Confidentiality
  - is the property that data or services are protected from unauthorized access.
  - This means that a hacker cannot access your income tax returns on a government computer.

Security Characteristics

- Integrity
  - is the property that data or services are being delivered as intended.
  - This means that your grade has not been changed since your instructor assigned it.

- Assurance
  - is the property that the parties to a transaction are who they purport to be.
  - This means that, when a customer sends a credit card number to an Internet merchant, the merchant is who the customer thinks they are.

Security Characteristics

- Availability
  - is the property that the system will be available for legitimate use.
  - This means that a denial-of-service attack won't prevent your ordering this book.

- Auditing
  - is the property that the system tracks activities within it at levels sufficient to reconstruct them.
  - This means that, if you transfer money out of one account to another account, in Switzerland, the system will maintain a record of that transfer.

Security Tactics

- Tactics for achieving security can be divided into
  - those concerned with resisting attacks,
  - those concerned with detecting attacks,
  - and those concerned with recovering from attacks.

- Using a familiar analogy,
  - putting a lock on your door is a form of resisting an attack,
  - having a motion sensor inside of your house is a form of detecting an attack,
  - and having insurance is a form of recovering from an attack.

Security Tactics-resisting attacks

- we identified
  - nonrepudiation, confidentiality, integrity, and assurance as goals in our security characterization.

- The following tactics can be used in combination to achieve these goals.
  - Authenticate users.
  - Authorize users.
  - Maintain data confidentiality.
    - Encryption
    - Communication links
      - virtual private network (VPN)
      - Secure Sockets Layer (SSL)
  - Maintain integrity.
    - checksums
    - hash results
  - Limit exposure
  - Limit access
    - Firewalls

Security Tactics-detecting attacks

- The detection of an attack is usually through an intrusion detection system.
  - Such systems work by comparing network traffic patterns to a database.
  - In the case of misuse detection, the traffic pattern is compared to historic patterns of known attacks.
  - In the case of anomaly detection, the traffic pattern is compared to a historical baseline of itself.
  - Frequently, the packets must be filtered in order to make comparisons.
  - Filtering can be on the basis of protocol, TCP flags, payload sizes, source or destination address, or port number.

- Intrusion detectors must have
  - some sort of sensor to detect attacks,
  - managers to do sensor fusion,
  - databases for storing events for later analysis,
  - tools for offline reporting and analysis,
  - and a control console so that the analyst can modify intrusion detection actions.

Security Tactics-recovering from attacks

- Tactics involved in recovering from an attack can be divided into
  - those concerned with restoring state and
  - those concerned with attacker identification.

- The tactics used in restoring the system or data to a correct state overlap with those used for availability
  - since they are both concerned with recovering a consistent state from an inconsistent state.

- The tactic for identifying an attacker is
  - to maintain an audit trail.

作业五

- 请你在大二开发的E-Book系统的基础上，完成下列任务：
  1. 下面两项任务你可以选择一项完成：
     ① 开发一个微服务，输入为书名，输出为书的作者。将此微服务单独部署，并使用netflix-zuul进行路由，在你的E-Book系统中使用该服务来完成作者搜索功能。
     ② 开发一个函数式服务，输入为订单中每种书的价格和数量，输出为订单的总价。将此函数式服务单独部署，并在你的E-Book系统中使用该服务来完成作者搜索功能。
  2. 在你的工程中增加HTTPS通信功能，并且观察程序运行时有什么不同。请你编写文档，将程序运行时的过程截图，并解释为什么会出现和之前不同的差异。

  – 请将你编写的相关代码整体压缩后上传，请勿压缩整个工程提交；
  – 关于第2点的文档一并压缩提交。

- 评分标准：
  1. 能够正确实现上述搜索/计费功能 。(3分)
  2. 能够正确记录并分析增加HTTPS后程序运行的方式。(2分)

References

- Core Java (volume II) 11th edition
  - http://horstmann.com/corejava.html
- Single Sign On,
  - http://en.wikipedia.org/wiki/Single_sign-on
- Kerberos: The Network Authentication Protocol,
  - http://web.mit.edu/kerberos/
- Kerberos(protocol),
  - http://en.wikipedia.org/wiki/Kerberos_(protocol)
- CAS
  - https://www.apereo.org/projects/cas
- **【CAS学习之一】CAS入门**
  - https://www.cnblogs.com/cac2020/p/13719609.html
- The Java EE 7 Tutorial
  - http://docs.oracle.com/javaee/7/tutorial/doc/javaeetutorial7.pdf
- Software Architecture in Practice, Second Edition
  - By Len Bass, Paul Clements, Rick Kazman
  - Publisher : Addison Wesley

References

- 如何在Spring启动时在Spring Security级别启用CORS(How to enable CORS at Spring Security level in Spring boot)
  - http://www.it1352.com/978249.html
- Securing a Web Application
  - https://spring.io/guides/gs/securing-web/
- SSL/TLS Configuration How-To
  - https://tomcat.apache.org/tomcat-9.0-doc/ssl-howto.html#Configuration
- spring boot进行开启SSL安全验证（application.properties不能配置两个端口）
  - https://blog.csdn.net/lan12334321234/article/details/84912188
- Springboot配置ssl证书踩坑记
  - https://blog.csdn.net/qq_16410733/article/details/89518650
- Tomcat8.5配置https和SpringBoot配置https
  - https://blog.csdn.net/wangxudongx/article/details/89534071

Thank You!