# PIV Card Authentication Documentation

**Documentation created by Ben Nordin and updated by Nicolas Crausaz**

This documentation goes through the steps to make Smart Card Authentication possible on a tomcat server. More specifically, this documentation will show how to authenticate clients with government PIV (Personal Identity Verification) cards. The application created will give two options for authentication. One is a regular username and password form, and the other is with an X.509 certificate. All required files (except for certificates) are also on the GitHub repository: https://github.com/nicolascrausaz/TomcatPIV

## Required Technologies

This authentication process can of course be done with a number of different technologies, but for this documentation, Tomcat will be used with Java Spring. Before beginning, the following technologies should be understood and prepared:

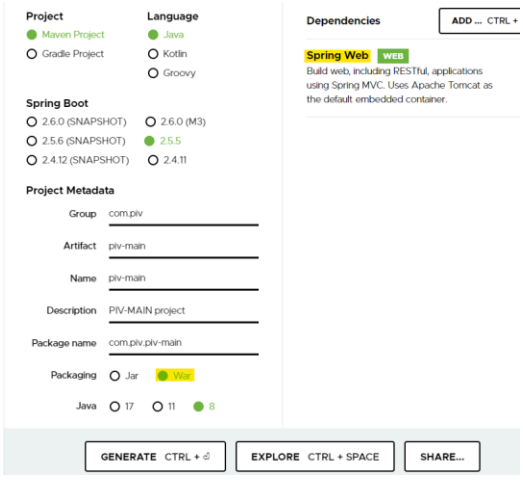
- Tomcat 9
- Java Spring
- Maven
- Keytool

## Prepare Java Spring Applications

In order to accomplish optional PIV Card authentication, two separate spring applications must be created and deployed onto Tomcat. The first application will contain everything except for the authentication with the PIV card. The second application will only contain PIV card authentication. Each application works with Spring MVC to accomplish their tasks.

## PIV-MAIN

1. Create a spring application on the initializer website (https://start.spring.io/ )
2. Choose the **war** package and add the **Spring Web** dependency



3. Generate the project code
4. Import project to IDE
5. Add the following dependencies into the pom.xml file

```
<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
	</dependency>

	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-test</artifactId>
		<scope>test</scope>
	</dependency>

	<dependency>
		<groupId>org.apache.tomcat.embed</groupId>
		<artifactId>tomcat-embed-jasper</artifactId>
		<scope>provided</scope>
	</dependency>

	<dependency>
		<groupId>javax.servlet</groupId>
		<artifactId>jstl</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-tomcat</artifactId>
		<scope>provided</scope>
	</dependency>
</dependencies>
```

6. The application will consist of two pages. One will be a login page where the user can select between PIV Card authentication and a regular username and password. The second page will output the username or CN of the logged in user. To do this, we will use JSP views. Place the following files inside the /src/main/webapp/WEB-INF/jsp You will need to create those directories.

**user.jsp**

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
<title>X.509 Authentication Demo</title>
</head>
<body>
	<h1>Login Successful!</h1>

	<form action="/piv-main/user">
		Username:
		<c:out value="${username}" />
		<br> <input type="submit" value="Back" />
	</form>
</body>
</html>
```

**userlogin.jsp**

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
<title>X.509 Authentication Demo</title>

</head>
<body>

	<h1>Login Page</h1>
	<form action="/piv-main/redirect" method="get">
		<input type="submit" value="PIV AUTH" />
	</form>
	<br>
	<form method="GET" action="/piv-main/userlogin">
		Username: <input type="text" name="username" /> Password: <input
			type="password" name="password" /> <br> <input type="submit"
			value="login" />
	</form>
</body>
</html>
```

7. Next, we need to add properties so that Tomcat knows where our .jsp files are. Open application.properties under src/main/resources and add the following lines.

```
# ==============================================
# VIEW RESOLVER
# ==============================================

spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

8. The next file is our controller. The controller contains four simple mappings.
   a. / and /user returns the view userlogin
   b. /userlogin is what the submit button on our userlogin JSP hits. This sends the username entered in the textbox to the controller, then redirects to /userpage
   c. /userpage returns the view user with the username added to the model
   d. /redirect redirects the user to the PIV Card authentication application

   PivController also has a String authUrl which is the URL where the piv authentication application is stored. In this tutorial, domain name can be localhost.

**PivController.java**

```
@Controller
public class PivController {
	String authUrl = "https://<domainname>:8443/piv-auth/";

	@GetMapping({"/","/user"})
	public String user() {
		return "userlogin";
	}

	@GetMapping("/userlogin")
	public String formLogin(String username, String password) {
		return "redirect:/userpage/" + username;
	}

	@GetMapping("userpage/{username}")
	public String userPage(@PathVariable String username, Model model) {
		model.addAttribute(username);
		return "user";
	}

	@GetMapping("/redirect")
	public void method(HttpServletResponse httpServletResponse) {
		httpServletResponse.setHeader("Location", authUrl);
		httpServletResponse.setStatus(302);
	}

}
```

9. Finally, we have the User POJO that stores the username and password of a user.

**User.java**

```
public class User {
	private String username;
	private String password;

	public User(String username, String password) {
		super();
		this.username = username;
		this.password = password;
	}

	public String getUsername() {
		return username;
	}

	public void setUsername(String username) {
		this.username = username;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}
}
```

10. Once all files are constructed, run the application to ensure that everything works. The links will not work due to the application running on the embedded tomcat server and not the standalone server that we will be deploying on, but try hitting "/" or "/user".
11. One more useful component to add to this application is a special plugin to the pom.xml file. This plugin makes it much easier to deploy our applications to our tomcat server. All that's needed is to run ***mvn compile war:exploded*** and maven will place our exploded war file into a location of our choosing. I chose to put it directly into my webapps folder inside of tomcat, but you can make it go wherever you'd like.

```
<plugin>
	<groupId>org.apache.maven.plugins</groupId>
	<artifactId>maven-war-plugin</artifactId>
	<configuration>
		<webappDirectory>C:\Program Files\apache-tomcat-9.0.54\webapps\piv-main
		</webappDirectory>
	</configuration>
</plugin>
```

## PIV-AUTH

1. Create a spring application on the initializer website (https://start.spring.io/ )
2. Choose the **war** package and add the **Spring Web** dependency
3. Generate the project code
4. Import project to IDE
5. Add the following dependencies into the pom.xml file

```
<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
	</dependency>

	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-test</artifactId>
		<scope>test</scope>
	</dependency>

	<dependency>
		<groupId>org.apache.tomcat.embed</groupId>
		<artifactId>tomcat-embed-jasper</artifactId>
		<scope>provided</scope>
	</dependency>

	<dependency>
		<groupId>javax.servlet</groupId>
		<artifactId>jstl</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-tomcat</artifactId>
		<scope>provided</scope>
	</dependency>
</dependencies>
```

6. This application only acts as a certificate extractor, therefore, the only file that we really need is the controller. All other configuration will happen in the tomcat settings. In this controller, the "/" endpoint will find the X.509 certificate provided by tomcat, then extract the CN or Common Name from the file. In this simple example, the controller will then redirect the user back to the main application with just the common name in hand. All of the certificate information is in the subjectDN variable, and this can be used as you wish. In this tutorial, domain name can be localhost.

**AuthController.java**

```
@Controller
public class AuthController {
	String projectUrl = "https://<domainname>:8443/piv-main/userpage/%s";

	@GetMapping(value = "/")
	public void auth(HttpServletResponse response) throws InvalidNameException {
		HttpServletRequest request = ((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes())
				.getRequest();
		X509Certificate[] certs = (X509Certificate[])
request.getAttribute("javax.servlet.request.X509Certificate");
		X509Certificate clientCert = certs[0];
		X500Principal subjectDN = clientCert.getSubjectX500Principal();
		LdapName ln = new LdapName(subjectDN.toString());
		String username = "";
		for (Rdn rdn : ln.getRdns()) {
			if (rdn.getType().equalsIgnoreCase("CN")) {
				username = (String) rdn.getValue();
				break;
			}
		}
		response.setHeader("Location", String.format(projectUrl, username));
		response.setStatus(302);
	}
}
```

7. This application cannot be fully tested quite yet because of the required tomcat configuration. If you run the project and hit the "/" endpoint, your server should throw a null pointer exception. This is because the application is trying to retrieve an X509 certificate, but tomcat has not provided one yet.

## Enable CAC authentication on Tomcat

This part of the documentation can be found in detail on the site (https://blog.e-zest.com/enable-tomcat-server-for-smart-card-authentication )

1. Create the key & cert for the Tomcat server - Go to any directory where you want to generate the keys and open command prompt and run following command. This command will create a .keystore file
   keytool -genkey -v -alias tomcat -keyalg RSA -sigalg SHA256withRSA -validity 365 -keystore .keystore -storepass password -keypass password -dname "CN=localhost, OU=orgUnit, O=org, L=fribourg, ST=fribourg, C=CH"

Note that the storepass and keypass has to be the same here. The CN should be the host/machine name that will appear in the HTTPS URL when accessing this Tomcat, so e.g. localhost.

2. Create the key & cert for client.
   keytool -genkey -v -alias clientKey -keyalg RSA -storetype PKCS12 -keystore clientKey.p12 -storepass password -keypass password -dname "CN=Nicolas Crausaz, OU=orgUnit, O=org, L=fribourg, ST=fribourg, C=CH"
3. Import the key file we have create in step 2. On Windows, you can double-click and import this *.p12 file into IE, or add it to Firefox via Tools / Options, Security, Certificates, View Certificates, Import. You'll have to type in the mypassword (above). BTW, again the storepass and keypass *HAS* to be the same here, else Windows/IE or Mozilla Certificate importing will fail.
4. Now we need to add the certificate (containing the public key) to the Tomcat keystore so that it recognizes this client certificate, by first exporting it from the keystore from step 3 and then importing it into the keystore from step 1.
   keytool -export -alias clientKey -keystore clientKey.p12 -storetype PKCS12 -storepass password -rfc -file clientKey.cer
5. Now import this exported certificate to the keystore we have created in first step
   keytool -import -v -file clientKey.cer -keystore .keystore -storepass password
6. Download the Certificate Authorities (CA) file of the authorities against which you have to verify the smart card. Digital certificates loaded into smart cards are issued against such Certificate Authorities file. For our example we have used PIVKey smart card, so we have to download it's server CA file from below link: http://ca.pivkey.com/
7. This file has to be imported into keystore file created in first step. To create a secure connection between client and server, tomcat server will demand for certificate and its private key (pin/password of the smart card) and will verify those credentials against this CA file.
   To import this certificate into keystore created in first step run below command
   keytool -importcert -file tagliotestca.crt -keystore .keystore -alias serverca
   After enter provide password mentioned in first step and then say "yes"

## Setting up the Tomcat Server

1. After downloading Tomcat 9 from https://tomcat.apache.org/download-90.cgi, locate the server.xml file in {CATALINE_BASE}/conf. In order to have X.509 authentication on just one of our applications, we must have two connectors. In this example, I use a non-SSL connector for the main application, and an SSL connector for the authentication application. The main application can be HTTP or HTTPS, but its required that the authentication application be HTTPS.

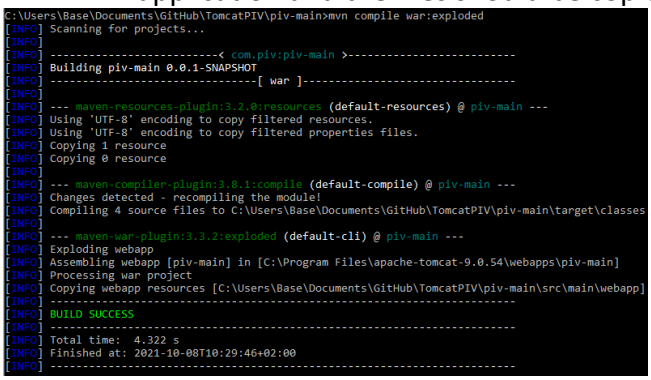
```
<Connector port="8080" protocol="HTTP/1.1"
       connectionTimeout="20000"
       redirectPort="8443" />


<Connector clientAuth="true"
port="8443"
scheme="https"
secure="true"
SSLEnabled="true"
keystoreFile="<your_directory>\.keystore"
keystorePass="password"
truststoreFile="<your_directory>\.keystore"
truststorePass="password"
SSLVerifyClient="require"
SSLEngine="on"
sslProtocol="TLS"/>
```

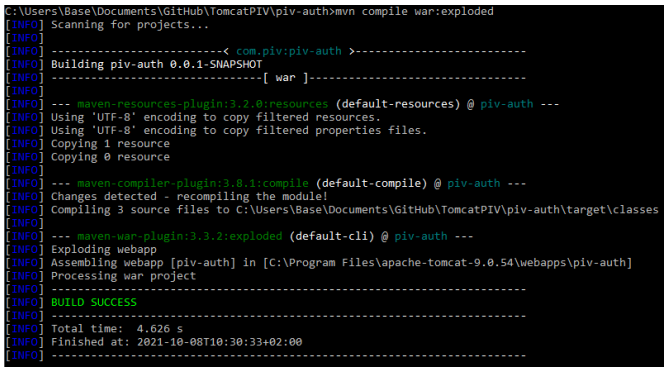
The first connector should already be created in the file. A few notable files:

1. Set the clientauth attribute to true (valid client certificate required for a connection to succeed) or want (use a certificate if available, but still connect if no certificate is available).
2. Tomcat reads these files from the CATALINA_BASE directory, not the /conf directory

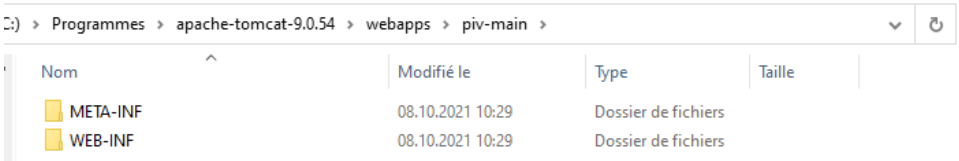
## Testing

1. The only thing left to do is to add the applications to Tomcat. With the plugin that we created earlier, we can call ***mvn compile war:exploded*** in both home directories of the application and the files should be copied directly over into /webapps.

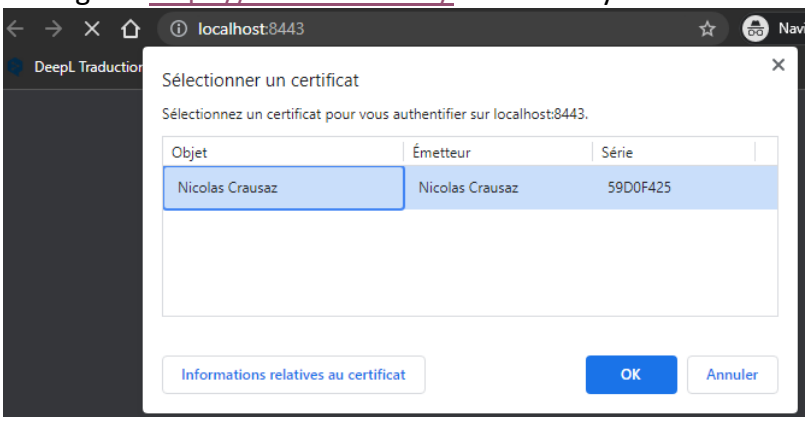



If everything worked well, you should have the war exploded in the webapps file of your Tomcat with a structure like this.  Of course the same for piv-auth.



2. To startup the server, navigate in a command prompt to the /bin folder and type in *startup*.
3. Then go to https://localhost:8443/ and select your certificate




4. Then go to https://localhost:8443/piv-main/


You can choose to login with regular login or with PIV Auth. If you select PIV AUTH you should have you certificate CN displayed. The one you have created in Create the key & cert for client. when generating the key.