# 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:

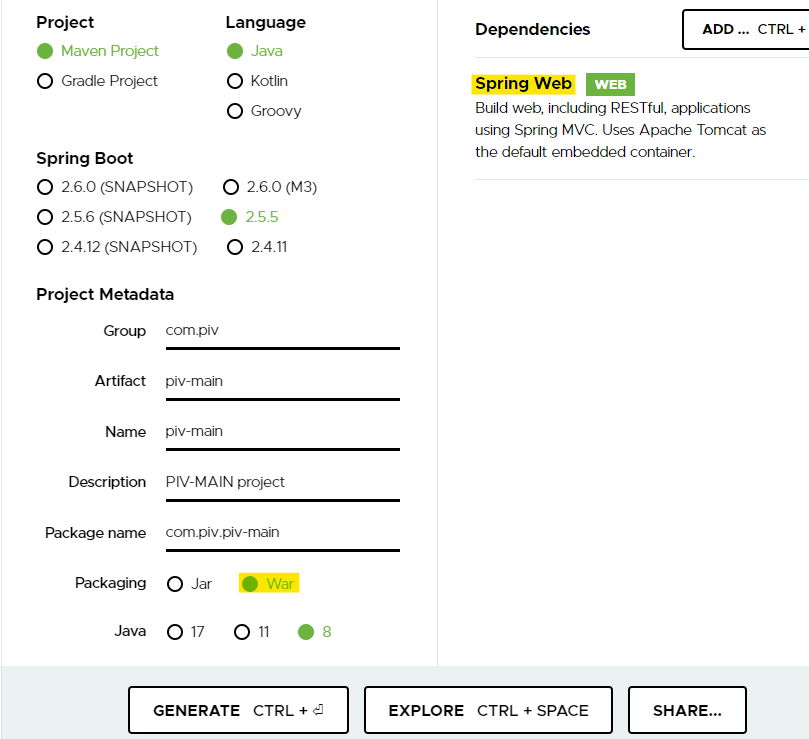
* Tomcat 9
* Java Spring
* Maven
* Keytool

## 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



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



1. 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**



**userlogin.jsp**



1. 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.



1. The next file is our controller. The controller contains four simple mappings.
2. / and /user returns the view userlogin
3. /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
4. /userpage returns the view user with the username added to the model
5. /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**



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

**User.java**



1. 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”.
2. 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.



## 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



1. 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**



1. 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=<domain-name>, 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.

1. Create the key & cert for client.

keytool -genkey -v -alias clientKey -keyalg RSA -storetype PKCS12 -keystore clientKey.p12 -storepass password -keypass password -dname "CN=<domain-name>, OU=orgUnit, O=org, L=fribourg, ST=fribourg, C=CH"

1. 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.
2. 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

1. Now import this exported certificate to the keystore we have created in first step

keytool -import -v -file clientKey.cer -keystore .keystore -storepass password

1. 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/> or from official government web site <http://repo.fpki.gov/fcpca/fcpcag2.crt>
2. 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 <certificatname>.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.

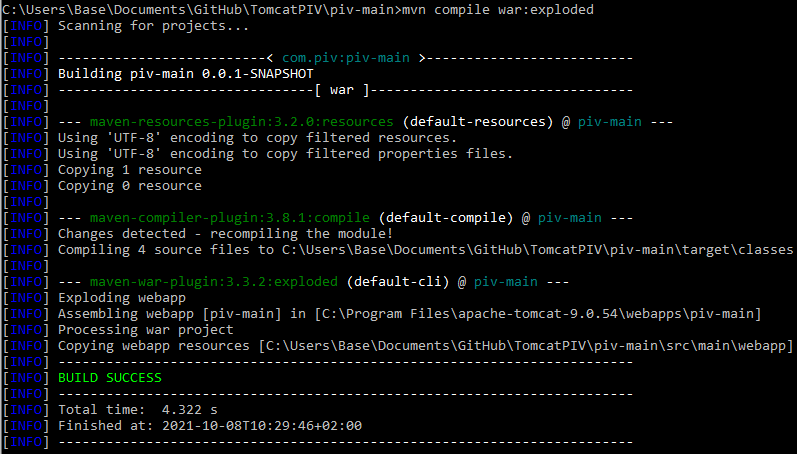
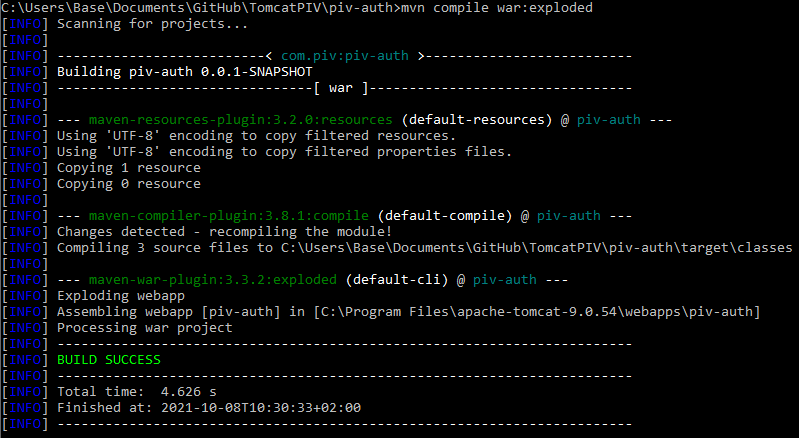


The first connector should already be created in the file. A few notable files:

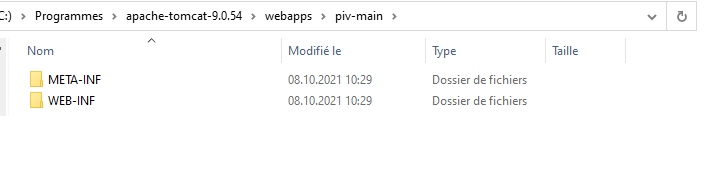
1. Set the clientauth attribute to true (valid client certificate required for a connection to succeed) or want (use a certificate if available, but still connect if no certificate is available).
2. Tomcat reads these files from the CATALINA\_BASE directory, not the /conf directory

## Testing

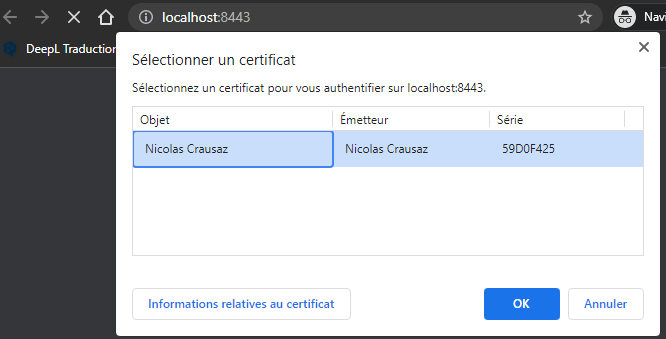
1. The only thing left to do is to add the applications to Tomcat. With the plugin that we created earlier, we can call ***mvn compile war:exploded*** in both home directories of the application and the files should be copied directly over into /webapps.

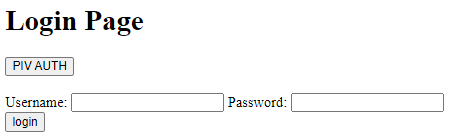
If everything worked well, you should have the war exploded in the webapps file of your Tomcat with a structure like this. Of course the same for piv-auth.



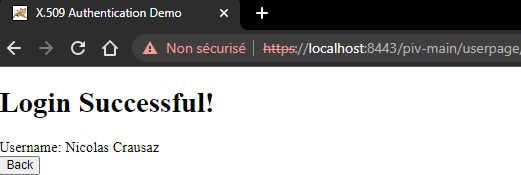
1. To startup the server, navigate in a command prompt to the /bin folder and type in *startup*.
2. Then go to <https://localhost:8443/> and select your certificate



1. Then go to <https://localhost:8443/piv-main/>

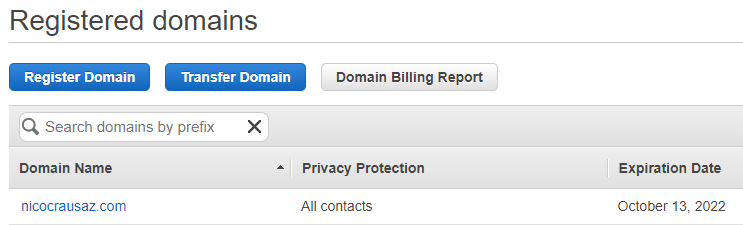


You can choose to login with regular login or with PIV Auth. If you select PIV AUTH you should have you certificate CN displayed. The one you have created in Create the key & cert for client. when generating the key.



## Deploy solution on AWS

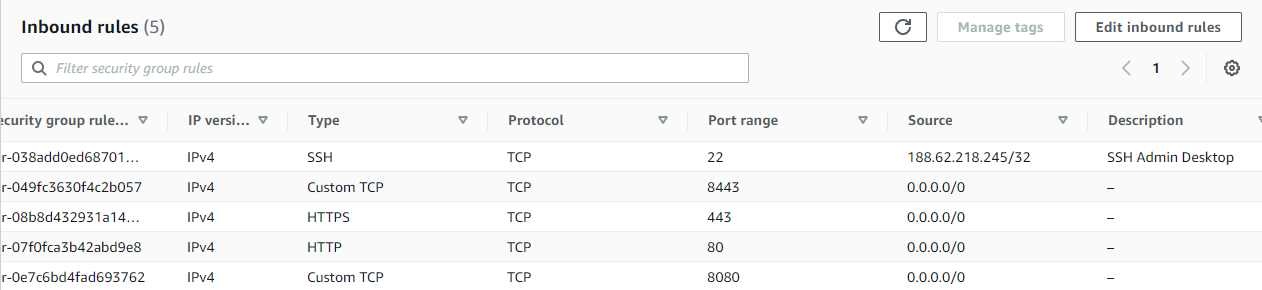
To deploy the solution on aws, it is necessary to have an aws domain available. You can buy a domain registered on the website <https://console.aws.amazon.com/route53/home#DomainRegistration>:



In this documentation, I bought the domain nicocrausaz.com The first thing we have to do is to change the URL at the beginning of our two controllers (PivController, AuthController) by the URL which contains this time our domain name

## Launch AWS EC2 instance

1. Log in to your AWS account and navigate to the EC2 console.
2. Select Instances, then Launch Instances.
3. For AMI, choose Ubuntu Server 20.04 LTS (HVM), SSD Volume Type.
4. You can choose from a wide array of instance types. I found that t2.micro (free tier) was not powerful enough to take a few simple requests so I upgraded to t2.medium and it has worked flawlessly thus far.
5. Select Next until you get to Step 6: Configure Security Group.
6. Select a premade security group or create a new one. The ports that we need to be open for incoming traffic are **8080**, **8443**, and **22**. 22 is so we can SSH into the instance. The source for the SSH is your private IP address.



1. Select Review and Launch, then Launch, and you will be prompted to select a key pair to SSH into the instance. Create a new one or select an existing pair. If you choose to create a new one, a .pem will download from your browser. Save this for later.
2. Select Launch Instances. This will take some time. The instance should show up under instances with the name “-“. I chose to name the instance “piv-test”
3. Finally, in the Instance Summary, under public IPv4 DNS, copy and save the shown address.

## Routing Domain Name to our Instance Using Route 53

The best way to route your domain name to your ec2 instance is by use of Route 53. If you are using a different service for your domain name, see their tutorials.

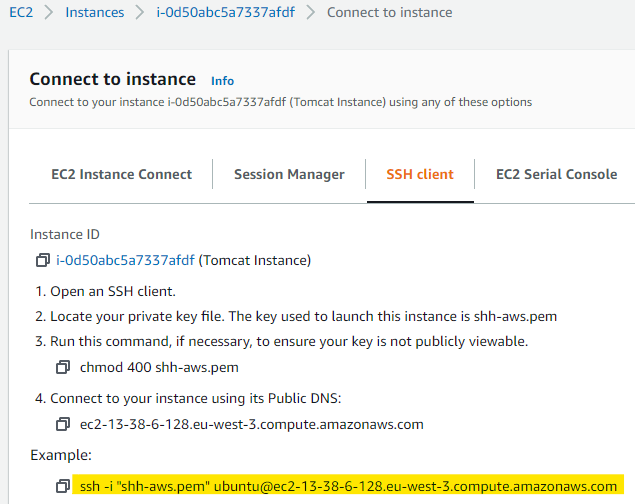
1. Navigate to Route 53 in AWS
2. Select Hosted zones, then the name of your domain.
3. Create a new Record Set.
4. For value, locate the Public IPv4 address (not the one from earlier) and paste it here.



After about 60 seconds, the AWS servers will update and the domain name will now be routed to your instance.

## Connecting to instance

Open a new terminal and copy the ssh command found under the "connect" tab of your instance. Make sure you insert the correct path for the .pem file you downloaded earlier. Sometimes it is necessary to change the permissions of the file to use it.



## Downloading Tomcat and Creating a Tomcat Service

Once inside our instance, we will need to download Tomcat and create a Tomcat Service. Because there are already great tutorials out there for doing this, I will just link the one I used: <https://www.digitalocean.com/community/tutorials/install-tomcat-9-ubuntu-1804> Steps 1 - 5 are all that are necessary for our uses.

## Deploy Files to our Instance

This is the final section to run our application. There are multiple ways to move our files from our local machine to our Ubuntu instance. One of the simplest ways is to push our files to a GitHub repository locally, then pull into our instance. This works especially well since git is already installed in our instance.

The files we want to deploy are the files generated by the command mvn compile war:exploded. You must not forget to copy the two folders piv-main and piv-auth which are in the webapps folder of your Tomcat. In my case, I copied these folders in the aws folder of my git directory. (Don’t forget to include fcpca.crt or whichever certificate CA you have choose and the .keystore we have generated)

Clone into your new repository:

*# sudo git clone* [*https://github.com/nicolascrausaz/TomcatPIV.git*](https://github.com/nicolascrausaz/TomcatPIV.git)

Copy files to required locations:

*# cp -r aws/piv-main /opt/tomcat/webapps*

*# cp -r aws/piv-auth /opt/tomcat/webapps*

*# cp aws/server.xml /opt/tomcat/conf*

*# cp aws/Certificat/.keystore /opt/tomcat*

*# cp aws/Certificat/fcpcag2.crt /opt/tomcat*

That’s it! Just run a quick *sudo systemctl restart tomcat* and you should be good to access your website ! Do not forget to add the port to the url.



The latest version of the authentication displays the subject and issuer of the certificate.

