Security through Digital Certificates – TLS Wed Application

A hand book on Application Security of java Application deployed in jboss

Basu, Sanjib

2017

Table of Contents

[1 Introduction 2](#_Toc499813814)

[1.1 Scope: 2](#_Toc499813815)

[1.2 Target Audience: 2](#_Toc499813816)

[2 Overview of SpRx Security: 2](#_Toc499813817)

[3 One way TLS Handshake - SpRx: 4](#_Toc499813818)

[3.1 Use Case: 4](#_Toc499813819)

[3.2 How TLS Handshake works: 4](#_Toc499813820)

[3.3 Configuration of Certificate: 5](#_Toc499813821)

[3.3.1 Steps to generate the Certificate: 11](#_Toc499813822)

[4 Two way TLS Handshake - SpRx: 11](#_Toc499813823)

[4.1 Use Case: 11](#_Toc499813824)

[4.2 How TLS Handshake works: 11](#_Toc499813825)

[4.3 Configuration of Certificate: 12](#_Toc499813826)

[4.3.1 Server side Certificates 12](#_Toc499813827)

[4.3.2 Client side Certificates 14](#_Toc499813828)

[5 Appendix: Glossary on TLS terminology 18](#_Toc499813829)

Table of Figures:

[Figure 1 . Sequence Diagram 1 – Security Design of Phase 1 Architecture - SpRx 3](#_Toc499813414)

[Figure 2 . Sequence Diagram 2 – Security Design of Phase 1 Architecture - SpRx 4](#_Toc499813415)

[Figure 3 . Server Certificate Diagram 1 5](#_Toc499813416)

[Figure 4 . Server Certificate Diagram 2 6](#_Toc499813417)

[Figure 5 . Server Certificate Diagram 3 6](#_Toc499813418)

[Figure 3 . Server Certificate Diagram 4 7](#_Toc499813419)

[Figure 7 . Production Server Certificate Diagram 1 8](#_Toc499813420)

[Figure 8 . Production Server Certificate Diagram 2 9](#_Toc499813421)

[Figure 9 . Production Server Certificate Diagram 3 9](#_Toc499813422)

[Figure 10 . Production Server Certificate Diagram 4 10](#_Toc499813423)

[Figure 11 . Intermediary CA Certificate Diagram 1 13](#_Toc499813424)

[Figure 12 . Intermediary CA Certificate Diagram 2 13](#_Toc499813425)

[Figure 13 . Intermediary CA Certificate Diagram 3 14](#_Toc499813426)

[Figure 14 . Client Certificate Diagram 1 15](#_Toc499813427)

[Figure 15 . Client Certificate Diagram 2 15](#_Toc499813428)

[Figure 16 . Client Certificate Diagram 3 16](#_Toc499813429)

# Introduction

The communication between a client browser, and server application, or between various components of application needs to be secured, in order to ensure data integrity in transit.

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide communications security over a computer network. The connection is secure because symmetric cryptography is used to encrypt the data transmitted. A digital certificate certifies the ownership of a public key by the named subject of the certificate, and indicates certain expected usages of that key. This allows others (relying parties) to rely upon signatures or on assertions made by the private key that corresponds to the certified public key.

## Scope:

This document outlines the design and approach for application security on digital certificates, in an application of leading Pharmacy company.

## Target Audience:

This document is a handbook for Application Developer, Security support, Designer, Architect and Manager.

# Overview of SpRx Security:

Following sequence diagrams outlines the security of application:



Figure . Sequence Diagram 1 – Security Design of Phase 1 Architecture



Figure . Sequence Diagram 2 – Security Design of Phase 1 Architecture

# One way TLS Handshake:

## Use Case:

When client browser calls application for the first time, TLS handshake happens, as well as validation of SPARCS-token (the role and user repository) and hash takes place.

## How TLS Handshake works:

The F5 load balancer contains in keystore its private key, public certificate and any certificate chain required for Intermediary CA, such as Symantec. During TLS handshake, the F5 presents its certificate to client JRE (Java Runtime Environment), which validates against JRE truststore ($JAVA\_HOME\jre\lib\security\cacerts) the issuer of certificate, and the hostname (domain name as subject alternative name).

## Configuration of Certificate:

For the TLS certification, mentioned above. F5 Server (at default port 443) is installed with a Certificate that looks like following in all non-prod environments (Dev, SIT, UAT, and TRAIN):

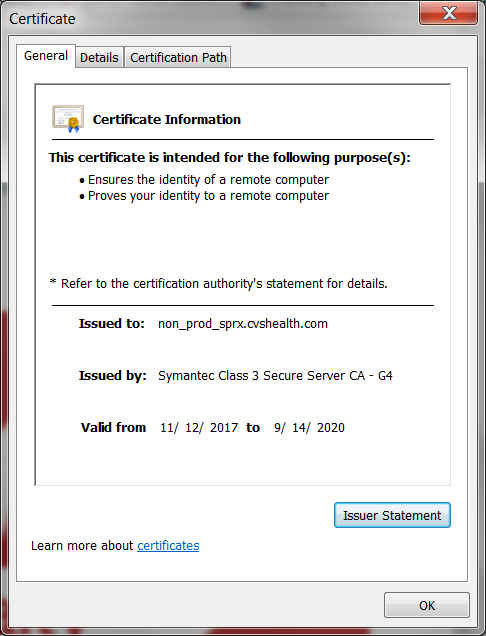


Figure . Server Certificate Diagram 1

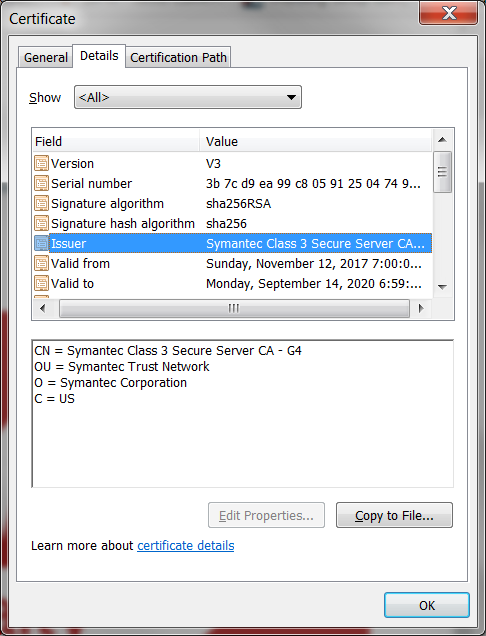


Figure . Server Certificate Diagram 2

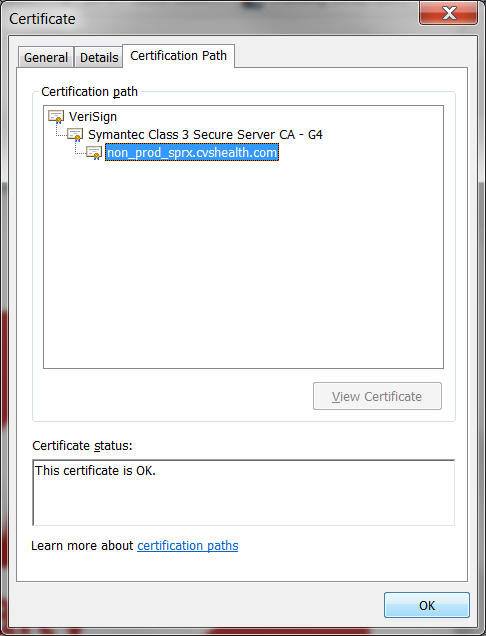


Figure . Server Certificate Diagram 3

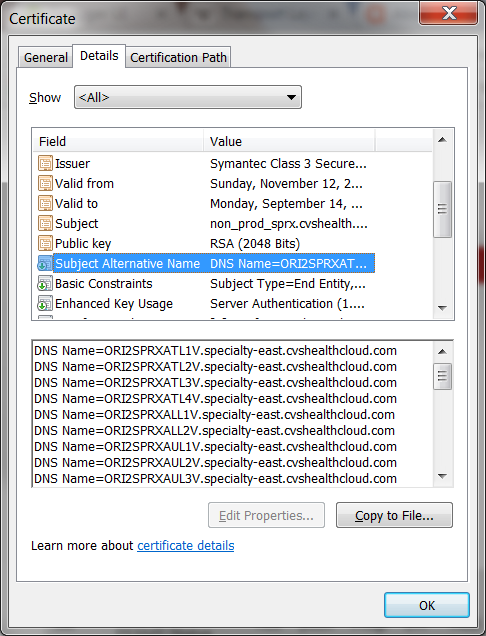


Figure . Server Certificate Diagram 4

Subject Alternative name contains all Web Servers, App Servers, and F5 VIP of Dev, SIT, UAT and TRAIN Environments of SpRx:

DNS Name=XXXX.specialty-east.cvshealthcloud.com

DNS Name=XXXX.specialty-east.cvshealthcloud.com

DNS Name=YYYY-dev1.cvshealth.com

DNS Name=YYYY-dev2.cvshealth.com

DNS Name=YYYY-sit1.cvshealth.com

DNS Name=YYYY-sit2.cvshealth.com

DNS Name=YYYY-uat1.cvshealth.com

DNS Name=YYYY-uat2.cvshealth.com

DNS Name=YYYY-train.cvshealth.com

DNS Name=non\_prod\_YYYY.cvshealth.com

Note: Mostly in CVS, the Root CA (Certification Authority) of the certificate is a Public CA such as Verisign. JRE library (truststore) of client normally contains various Public CA cert by default installation, so that it can identify the issuer of the certificate. The private certificates, such as root-signed by intermediary CA (e.g. Symantec), are rarely used. By the contract with Symantec, there was no cost difference between public and private certificate.

In PROD and PT Environment, F5 Server (at default port 443) is installed with a Certificate that looks like following:

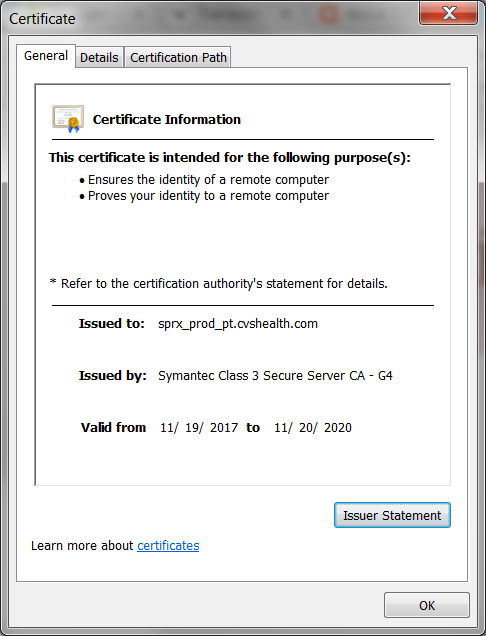


Figure . Production Server Certificate Diagram 1

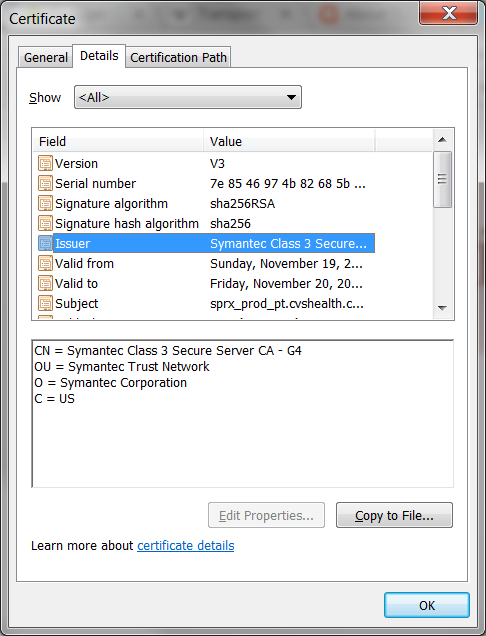


Figure . Production Server Certificate Diagram 2

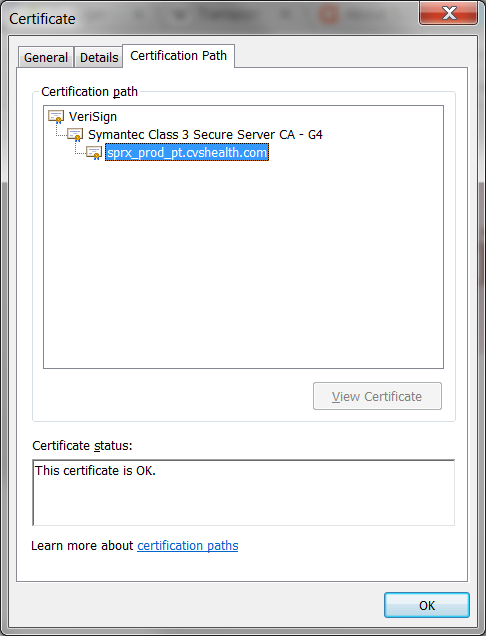


Figure . Production Server Certificate Diagram 3

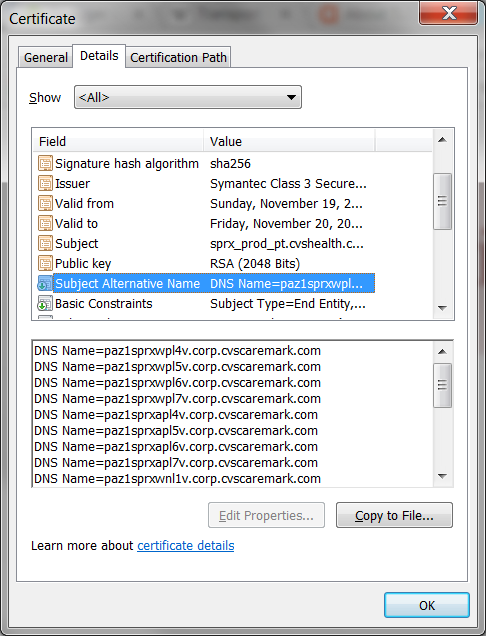


Figure . Production Server Certificate Diagram 4

Subject Alternative name contains all Web Servers, App Servers, and F5 VIP of PT and PROD Environments:

DNS Name=XXXX.corp.cvscaremark.com

DNS Name=XXXX.corp.cvscaremark.com

DNS Name=YYYY-pt.cvshealth.com

DNS Name=YYYY-prod.cvshealth.com

DNS Name=YYYY\_prod\_pt.cvshealth.com

### Steps to generate the Certificate:

#### 

1. Please provide the list of domain names (sever names) in order to generate a server Cert to the infrastructure Security team.
2. Infrastructure support users a tool to generate Public Certificate and provide in the PFX format, viz. file name is non\_prod\_YYYY.cvshealth.com.pfx. The PFX file is password protected.
3. The pfx file contains the private key, public certificate and the intermediary CA chain. Following is the listing of an example pfx:

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

Keystore type: JKS

Keystore provider: SUN

Your keystore contains 1 entry

non\_prod\_sprx.cvshealth.com, Nov 29, 2017, PrivateKeyEntry,

Certificate fingerprint (SHA1): 88:43:B9:65:D7:72:00:26:B9:76:34:05:45:FA:76:4F:69:FC:C8:A1

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

1. The pfx file is then provided to the F5 support team to install in the default port 443 with the password.

# Two way TLS Handshake:

## Use Case:

When a Java Batch Program wants to access Java API, two way TLS authentication takes place, as well as user role is validated.

In “Figure 1 . Sequence Diagram 1 – Security Design of Phase 1 Architecture - SpRx . Sequence Diagram 1”, the client application flow represents this scenario.

## How TLS Handshake works:

1. The F5 load balancer has pass through port 8443 which allows the HTTPS call to one of the Web Server, which redirects to one of the JBoss App Server.
2. The JBoss server contains in keystore its private key, public certificate and any certificate chain required for Intermediary CA, such as Symantec.
3. During TLS handshake, the JBoss presents its certificate to client JRE (Java Runtime Environment), which validates against JRE truststore ($JAVA\_HOME\jre\lib\security\cacerts) the issuer of certificate, and the hostname (domain name as subject alternative name).
4. Client application contains in keystore its private key, and public key. The any certificate chain required for Intermediary CA of client certificate, such as the Pharmacy company, is stored in JBoss server’s truststore.
5. During TLS handshake, the client presents its certificate to the JBoss Server, which validate the issuer of certificate against its truststore.

## Configuration of Certificate:

### Server side Certificates

The F5 certificate configuration is exactly same as described in “3.3 Configuration of Certificate:”.

Additionally, following steps are required:

1. The same “non\_prod\_YYYY.cvshealth.com.pfx” keystore is converted into JKS format “non\_prod\_YYYY.cvshealth.com.jks” and installed in the Jboss Server’s HttpsRealm.

=== JBoss CLI Command ============

/host=slave1/core-service=management/security-realm=HTTPSRealm/**server-identity**= ssl:add(keystore-path=${jboss.home.dir}/bin/keystore\_dev.jks, keystore-password= ${VAULT::CERT\_PWD::SERVER\_CERT::1}, alias=non\_prod\_YYYY.cvshealth.com)

1. The intermediary (CVS Pharmacy) CA cert chain is imported into a truststore with a password, and installed in the Jboss Server’s HttpsRealm

=== JBoss CLI Command ============

/host=slave1/core-service=management/security-realm=HTTPSRealm/**authentication**=truststore:add(keystore-path=/appl/sprx/domain/keys/truststore\_client\_dev.jks, keystore-password=${VAULT::CERT\_PWD::SERVER\_CERT::1})

The intermediary CA certificate chain looks like following:

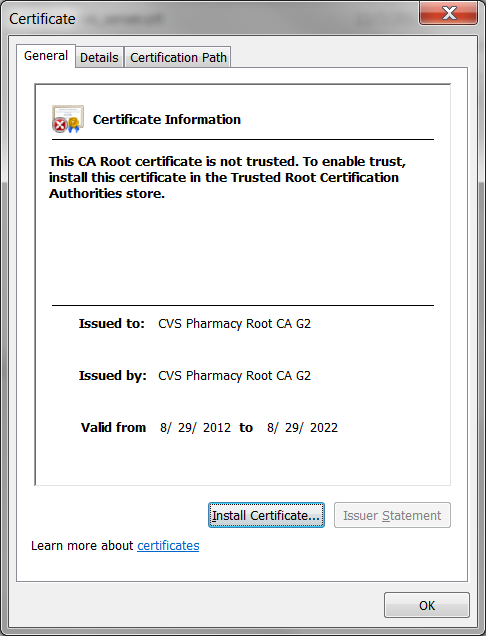


Figure . Intermediary CA Certificate Diagram 1

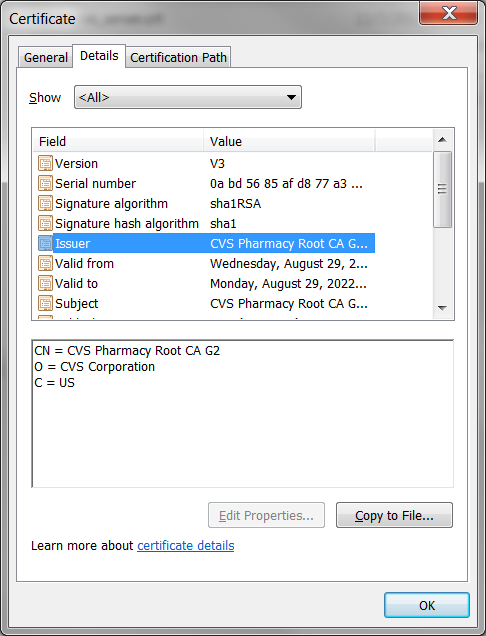


Figure . Intermediary CA Certificate Diagram 2

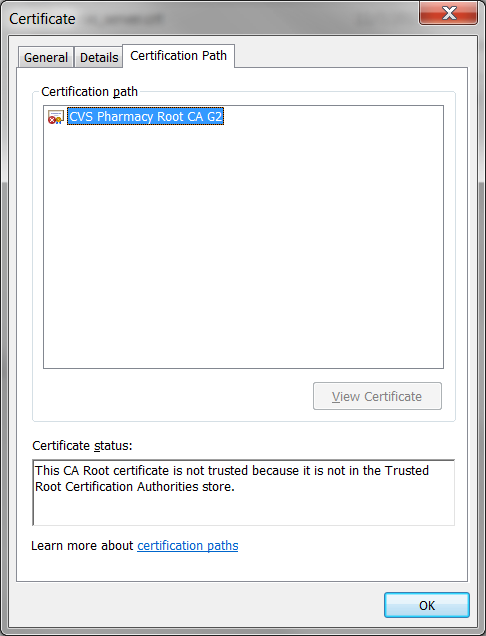


Figure . Intermediary CA Certificate Diagram 3

### Client side Certificates

The client side certificate needs to be signed by CA (intermediary CA such as the Pharmacy Company). An example certificate looks like following:

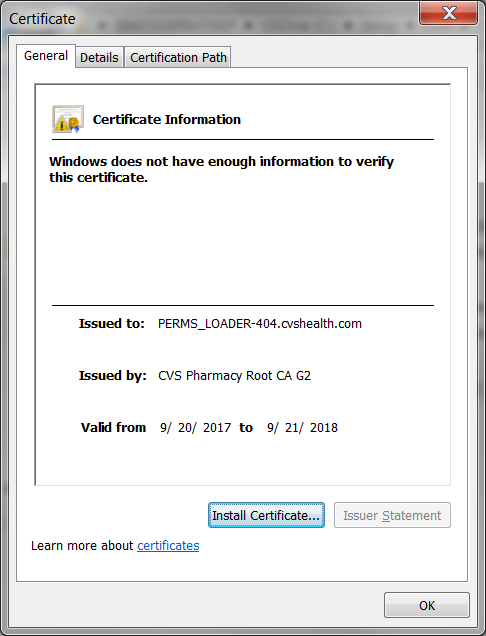


Figure . Client Certificate Diagram 1

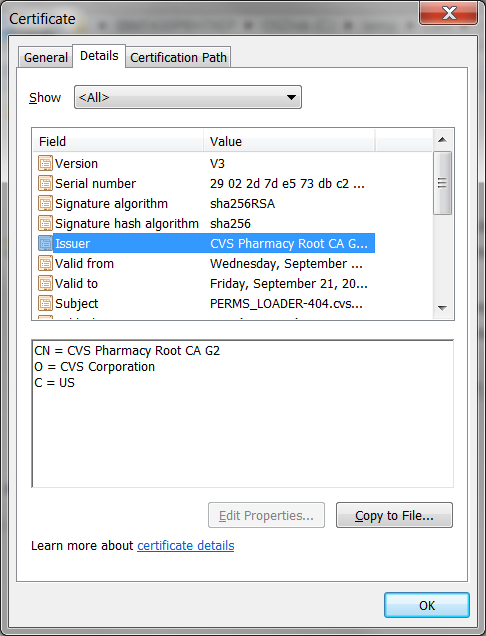


Figure . Client Certificate Diagram 2

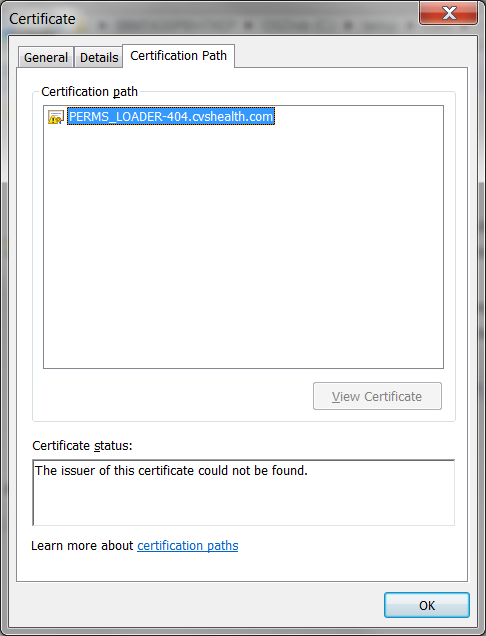


Figure . Client Certificate Diagram 3

Note: In PERMS standalone Batch client (PERMS\_LOADER), each environment (Dev, SIT, UAT, and PROD) had separate client certificate, because CN Subject Name contained a Role ID number unique for the Environment, which was later used to generate JWT token. PROD and PT Environments had same client certificate.

#### Steps to generate the Client Certificate:

1. Client application support generates a private key with Subject name of client.
2. Client application support generates the CSR (Certificate Signing Request) and send to the CVS infrastructure team.
3. Infrastructure team provides the signed client certificate along with intermediary CA cert chain of the signer.
4. Client application support creates client keystore including private key and the signed public Certificate. Following example is listing of a Client Keystore:

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

Keystore type: JKS

Keystore provider: SUN

Your keystore contains 1 entry

perms\_loader-404.cvshealth.com, Nov 13, 2017, PrivateKeyEntry,

Certificate fingerprint (SHA1): A0:20:33:BE:B2:07:90:29:E6:99:A3:16:CB:F2:51:31:19:88:66:77

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

1. In client application, use the keystore such as “client\_keystore\_dev.jks”.
2. The intermediary (CVS Pharmacy) CA cert chain is imported into a truststore with a password, and installed in the Jboss Server’s HttpsRealm

=== JBoss CLI Command ============

/host=slave1/core-service=management/security-realm=HTTPSRealm/**authentication**=truststore:add(keystore-path=/appl/sprx/domain/keys/truststore\_client\_dev.jks, keystore-password=${VAULT::CERT\_PWD::SERVER\_CERT::1})

# Appendix: Glossary on TLS terminology

Some important terms and concepts for configuring TLS:

| **Term** | **Definition** |
| --- | --- |
| **TLS certificate** | A digital file that identifies an entity in a TLS transaction. A certificate, or *cert*, can be used to identify the TLS server and TLS client, depending on the TLS configuration. |
| **Self Signed Certificate** | Certificate that is not signed by a trusted CA. The issuer and the subject are identical; they are signed with the private key matching the public key they contain. |
| **Certificate chain** | Often one will not have a certificate signed by CA's root private key. Instead, one have one’s cert along with one or more intermediate certs that form a chain. The last intermediate cert in the chain is typically signed by the CA's root private key. |
| **CertificateAuthority** **(CA)** | A trusted entity, such as Symantec or VeriSign, used to issue certs and to validate the authenticity of a cert. One type of cert, called a *self- signed* cert, does not require a CA. |
| **Public key** | Used to encrypt data sent from a TLS client to a TLS server. The public key is included in the cert. All TLS clients have a copy of the server's public key. |
| **Private key** | Used on the TLS server to decrypt data. Only the TLS server has the private key - it is not shared with TLS clients. |
| **CSR** | A Certificate Signing Request (CSR) is a file generated on the TLS server based on the private key. The CSR contains the public key and other information like organization's name, location, and domain name. The CA will sign the CSR to create a TLS certificate. One typically generate a CSR, when one have an expired cert and want to renew it. |
| **Keystore** | Contains the TLS certificate and private key used to identify the entity during TLS handshaking. |
| **Truststore** | Contains root certificates for well-known public certificate authorities (CA), or root-chain for intermediate CA.  The standard Oracle Java JDK distribution includes a default truststore (cacerts) that contains root certificates for many well-known CAs. |
| **PEM file** | Text files which comply with the X.509 format for for storing a certificate, certificate chain, or private key. If your cert or private key is not defined by a PEM file, you can convert it to a PEM file by using utilities such as openssl. |
| **PKCS12/PFX file** | A binary file format for storing a certificate, any intermediate certificates, and the private key in a single encryptable file. PFX files usually have the extensions .pfx and .p12. PFX files are typically used on Windows machines to import and export certificates and private keys. |
| **Server Name Indication (SNI)** | Allows multiple HTTPS targets to be served off the same IP address and port without requiring those targets to use the same certificate. |