Cryptographic stores in Alfresco

Brown Bag

Angel Borroy
Search Team
Nov 5, 2020

Alfresco®
simply a better way™

Cryptographic Stores in Alfresco

Java KeyStores

In Theory

- Electronic Certificates
- Chain of Trust
- Public and Private CAs
- Cryptographic Stores
- mTLS Protocol

In Practice

- When to use mTLS Communication
- Cryptographic Tools
- Alfresco KeyStores
- Alfresco mTLS Configuration
- Using Custom Certificates

In Panic

- Troubleshooting

In Theory

Electronic Certificates

X509 Certificate

Issuer Name
DN

Common Name
CN

RSA 1024 bits with SHA 256

```
$ openssl x509 -in Alfresco_Client_Alfresco_CA.pem -text –noout

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 4097 (0x1001)
  Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=GB, ST=UK, L=Maidenhead, O=Alfresco Software Ltd., OU=Unknown, CN=Custom Alfresco CA
    Validity
      Not Before: Jun 30 09:24:08 2020 GMT
      Not After : Jun 28 09:24:08 2030 GMT
    Subject: C=GB, ST=UK, O=Alfresco Software Ltd., OU=Unknown, CN=Custom Alfresco Repository Client
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
        Public-Key: (1024 bit)
        Modulus:
          00:a2:89:cf:ff:8d:0f:f6:47:76:fd:66:5b:f5:b6:
          d8:26:9f:59:b1:3d:38:39:fa:7d:38:5e:0a:61:5e:
          5c:dd:e5:50:c2:1c:4d:99:db:26:de:f2:3b:26:47:
          5c:d1:8a:f6:e1:a5:04:ec:7c:60:3b:2a:5c:e3:7e:
          97:26:59:3a:ed:d7:da:69:c0:9e:47:5b:a0:03:64:
          73:29:35:70:70:e7:da:a4:b7:5a:c5:a5:08:52:9b:
          e7:95:72:7e:0d:a4:dd:b6:85:84:e7:c5:4c:7c:fc:
          89:93:de:88:f9:c7:5b:52:1f:59:95:04:89:3a:96:
          b9:e6:a0:e9:e3:d4:68:3a:87
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Basic Constraints:
        CA:FALSE
      Netscape Cert Type:
        SSL Server
      Netscape Comment:
        OpenSSL Generated Server Certificate
      X509v3 Subject Key Identifier:
        84:E1:8B:E1:3C:9E:66:20:79:8F:AE:C5:9E:06:50:23:F2:54:A1:72
      X509v3 Authority Key Identifier:
        keyid:2D:AC:E1:41:70:08:36:16:3F:E5:C9:A8:0C:B1:CF:CF:6B:A4:80:BC
        DirName:/C=GB/ST=UK/L=Maidenhead/O=Alfresco Software Ltd./OU=Unknown/CN=Custom Alfresco CA
        serial:94:78:32:24:4F:A5:07:2B
      X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
      X509v3 Extended Key Usage:
        TLS Web Server Authentication
      X509v3 Subject Alternative Name:
        DNS:localhost
  Signature Algorithm: sha256WithRSAEncryption
       14:4d:81:49:ca:e7:00:13:2e:74:1b:2a:de:41:a5:45:79:45:
       3c:1c:0b:58:30:a8:a0:a4:f2:52:36:ba:6c:e8:9b:7e:4c:15:
       8c:86:56:a4:e7:38:0d:13:e5:f3:d1:23:5f:f1:28:d8:d7:d6:
       6a:8c:9:21:ec:aa:9f:7d:4e:79:87:14:b7:d5:8f:e8:cc:67:
       2c:1b:84:fd:de:ef:ab:c2:49:e4:8f:9e:a4:2e:49:ef:75:79:
       c7:7b:e2:a9:16:c6:14:94:2a:70:9e:1e:82:d8:d7:c5:54:b5:
       3c:bb:17:00:e1:86:5f:5c:c7:fe:da:12:35:6f:33:55:ca:11
```

Dates valid

Private Key

Public Key

Keystore

Truststore

This should match with Server DNS Name

Distinguished Name
DN

Key Usage
Policies

Issuer Signature

Electronic Certificates: File Format

.pem – Base64 encoded DER certificate, *password*

.cer, .crt, .der – Binary DER form, *password*

.p7b, .p7c – Base 64 Ascii file with PKCS#7, just for *public* certificate(s) or CRL(s)

.p12 – PKCS#12, may contain certificate(s) (public) and *private* keys, binary format (ASN.1), *password*

.pfx – PFX, predecessor of PKCS#12 (usually contains data in PKCS#12 format, e.g., with PFX files generated in IIS)

Public and Private CAs

CA (Certificate Authority) is an entity that *issues* electronic certificates.

Public CA

- Trusted Third-Party for general public, mainly oriented to final *users*
- Issued certificates are trusted *by default* in Operating Systems and Browsers
- The information and services we provide on these servers is open in *Internet*

Private CA

- Trusted Third-Party for internal users and services
- Issued certificates aren't trusted by default, so you need to configure computers and servers in order to trust them
- The information and services we provide on these servers is restricted to *Intranet*

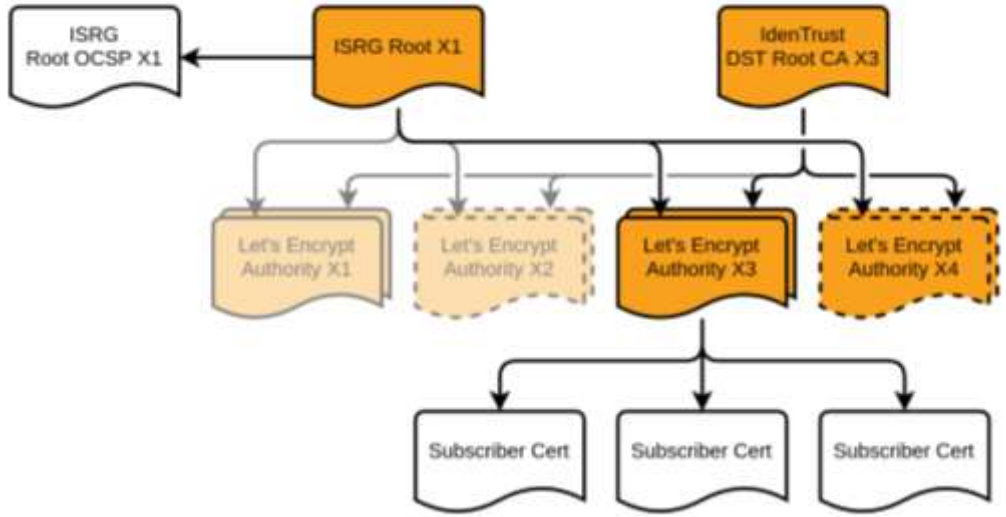
Chain of Trust

A certificate must be traceable back to the trust root it was signed with.

All **public certificates** in the chain [server, intermediate(s), and root] need to be present in the truststore.

- **Root Certificate:** A root certificate is a digital certificate that belongs to the issuing *Certificate Authority*.
- **Intermediate Certificate(s):** Intermediate certificates branch of root certificates like branches of trees. They act as middle-men between the protected root certificates and the server certificates issued.
- **Server Certificate** – The server certificate is the one issued to the specific server

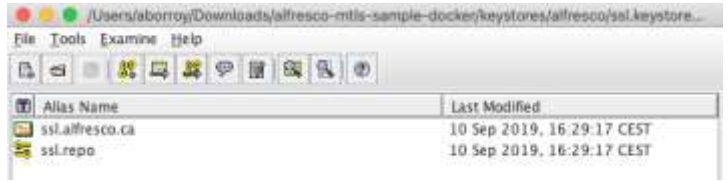
```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCiic//jQv2R3b9Zlv1ttgmn1mxPVg5+n04XgphXlzd5VDCHA2Z
...
nD6OWE6wMqGqCkzz/QlGPaR4n3E4cnm8YgsCZJRwZ/Q=
-----END RSA PRIVATE KEY-----

-----BEGIN CERTIFICATE-----
MIID2DCCA0GgAwIBAgICEAEwDQYJKoZIhvcNAQELBQAwfzELMAkGA1UEBhMCR0Ix
...
nh6C2NfFVLUwuxcA4YZfXMf+2hI1bzNVyhEZCQ==
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIC3DCCAkWgAwIBAgIJAJR4MiROpQcrMA0GCSqGSIb3DQEBCwUAMH8xCzAJBgNV
...
19vwF3KrjH0SGi8dEgF8iQ==
-----END CERTIFICATE-----
```

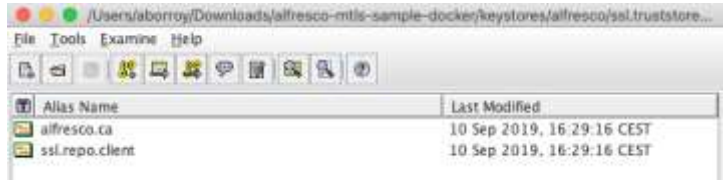


Cryptographic Stores

Java KeyStores are used to store key material and associated certificates.

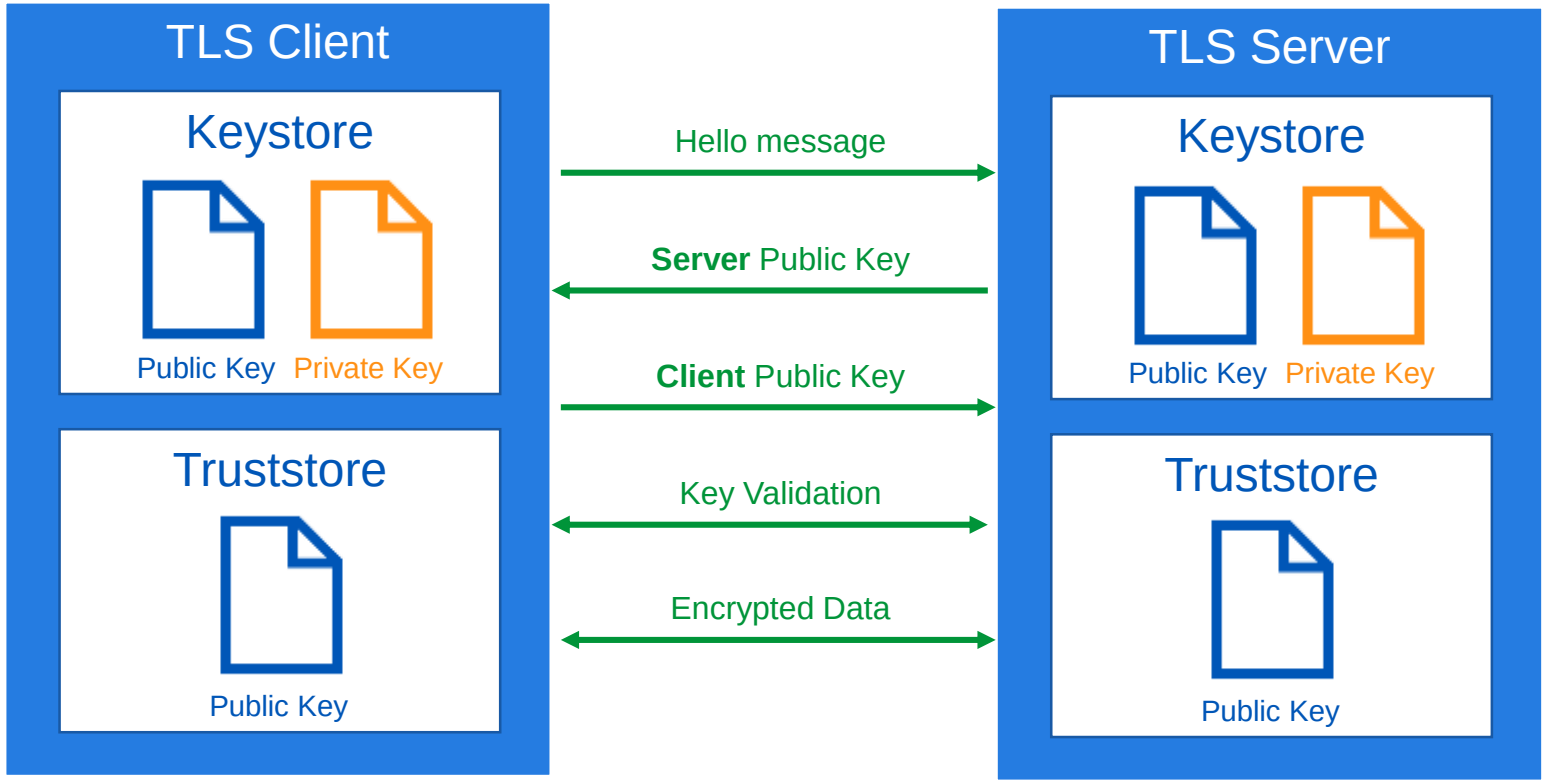
- Each key store has an *overall password* used to protect the entire store, and can optionally have *per-entry passwords* for each secret- or private-key entry.
- **Java Key Store (JKS)**
  - The original Sun JKS (Java Key Store) format is a proprietary binary format file that can only store asymmetric private keys and associated X.509 certificates.
- **JCE Key Store (JCEKS)**
  - Sun later updated the cryptographic capabilities of the JVM with the Java Cryptography Extensions (JCE). With this they also introduced a new proprietary key store format: JCEKS.
- **PKCS#12**
  - Apart from these proprietary key stores, Java also supports standard PKCS#12 format





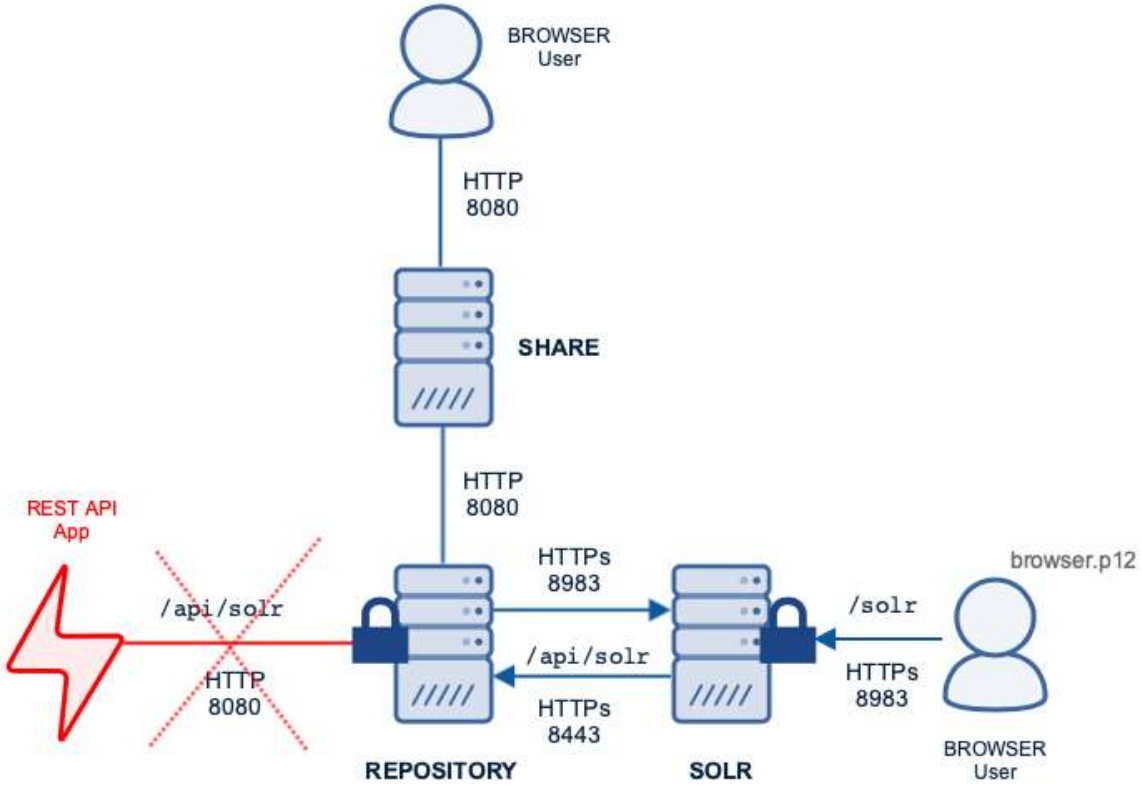
>> In **Alfresco** both “keystore” and “truststore” file types are Java Keystores stored in one of the formats described above (JKS, JCEKS, PKCS12)

mTLS Protocol



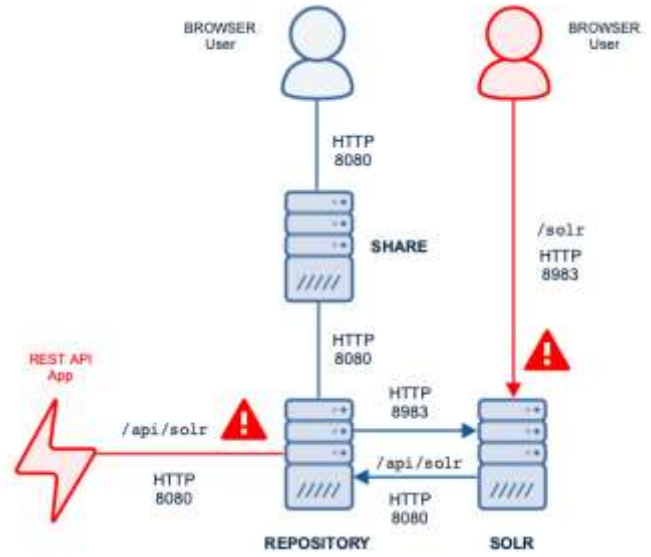
In Practice

When to use mTLS Communication

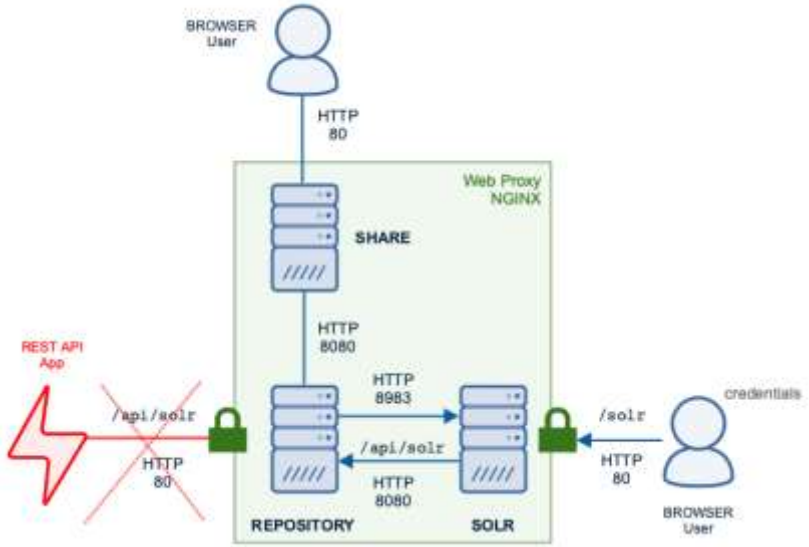


HTTPS protected with mTLS

HTTP default INSECURE



HTTP protected with pass



https://hub.alfresco.com/t5/alfresco-content-services-blog/alfresco-6-1-is-coming-with-mutual-tls-authentication-by-default/ba-p/287905

Cryptographic Tools

Issuing certificates

- *keytool* only supports self-signed certificates and a limited set of policies
- *openssl* allows to create an internal CA and to issue certificates signed by this CA with a full set of policies

Managing Certificates and Java KeyStores

- Command line
  - *keytool* provides the ability to create Java Keystores (JKS, JCEKS, PKCS12) including public and private certificates
- Window based programs (*keytool wrappers*)


Portecle


KeyStore Explorer

https://docs.oracle.com/en/java/javase/11/tools/keytool.html
https://www.openssl.org/docs/
http://portecle.sourceforge.net
https://keystore-explorer.org/index.html

Alfresco KeyStores: Repository

https://github.com/Alfresco/alfresco-ssl-generator

By default all the KeyStores are stored in JCEKS format

KeyStore and private certificates are protected by password

The alias (ssl.repo and so on) are not relevant, different ones can be used

keystore

- Not related with mTLS configuration, but with encrypting secrets*

ssl.keystore

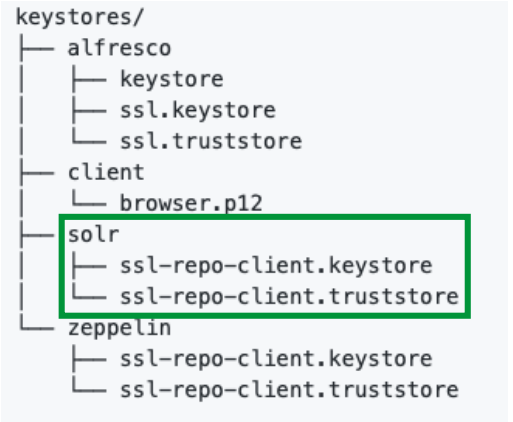
- ssl.repo is the private key used to sign HTTP requests
- ssl.alfresco.ca is the public key of the CA issuing the certificates

ssl.truststore

- alfresco.ca is the public key of the CA issuing the certificates
- ssl.repo.client is the public key of the certificate used by SOLR as client

* https://docs.alfresco.com/6.2/concepts/alf-keystores.html

Alfresco KeyStores: SOLR



https://github.com/Alfresco/alfresco-ssl-generator

By default all the KeyStores are stored in JCEKS format

KeyStore and private certificates are protected by password

The alias (ssl.repo and so on) are not relevant, different ones can be used

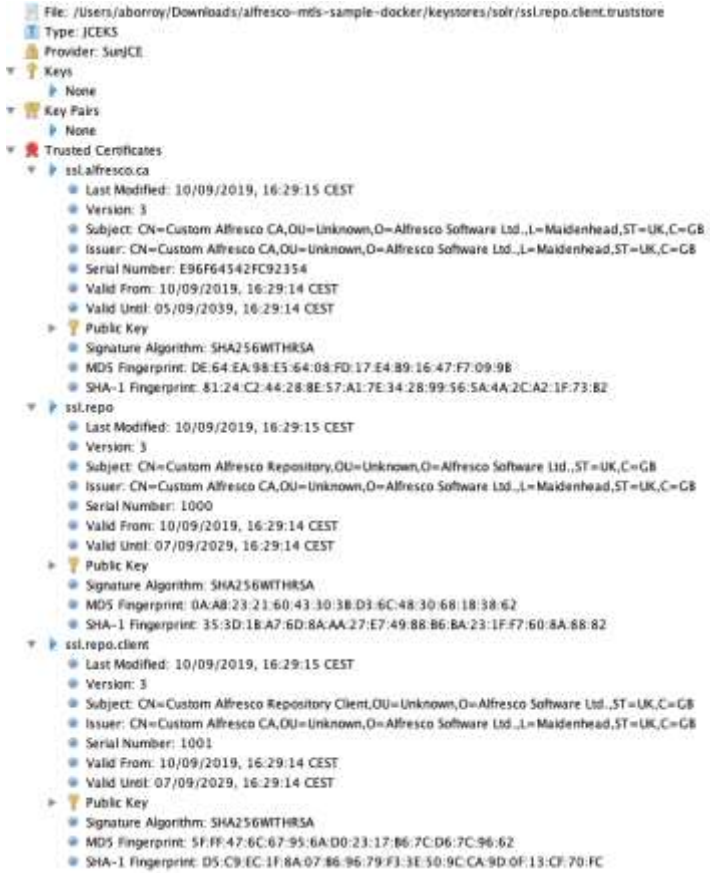
ssl-repo-client.keystore

- ssl.repo.client is the private key used to sign HTTP requests
- alfresco.ca is the public key of the CA issuing the certificates

ssl-repo-client.truststore

- ssl.alfresco.ca is the public key of the CA issuing the certificates
- ssl.repo is the public key of the certificate used by Repository as client
- ssl.repo.client is the public key of the certificate used by SOLR as client

>> Zeppelin is connecting with the Alfresco Repository, so the KeyStores are the same from SOLR



Alfresco KeyStores: Browser

https://github.com/Alfresco/alfresco-ssl-generator

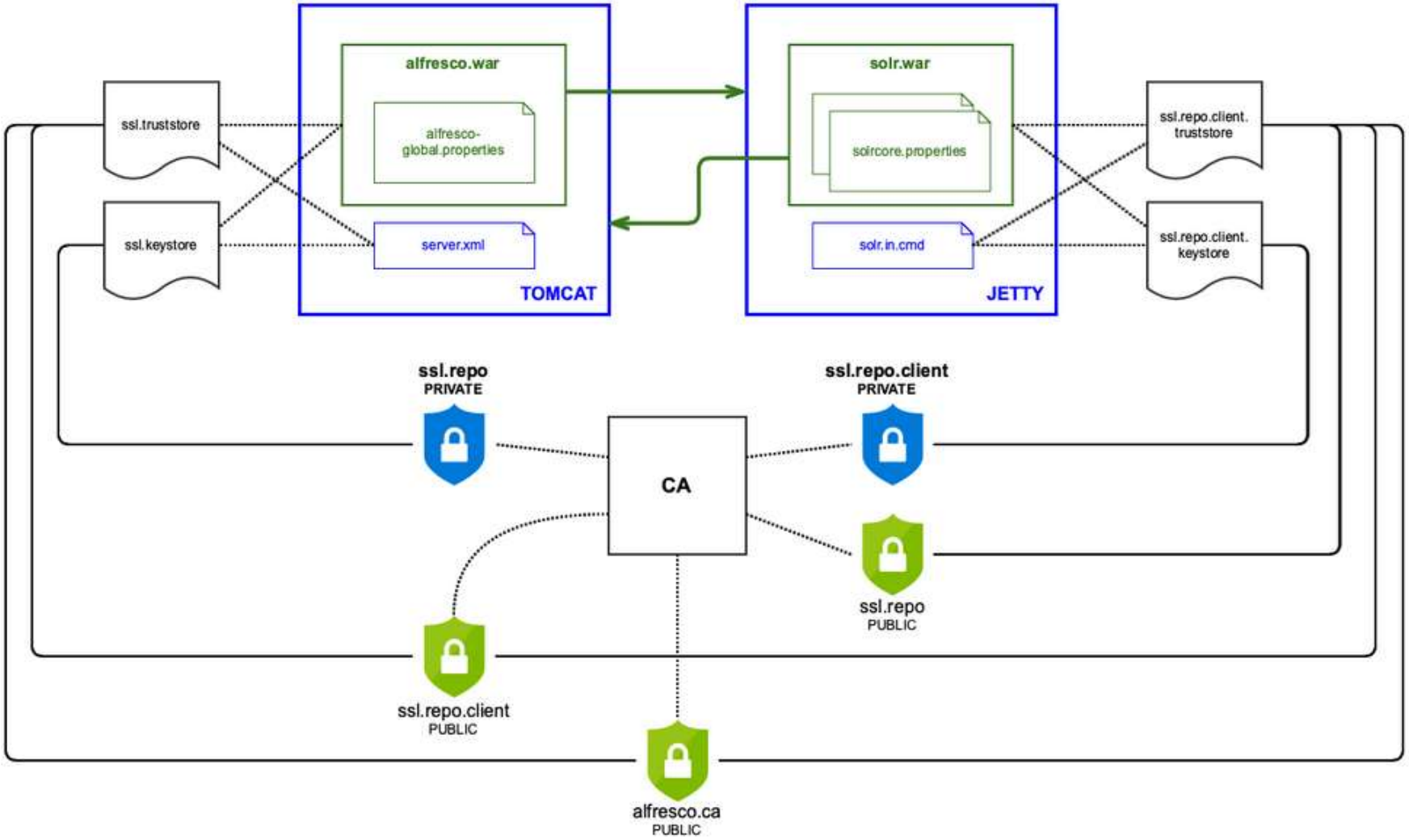
```
keystores/
├── alfresco
│   ├── keystore
│   ├── ssl.keystore
│   └── ssl.truststore
├── client
│   └── browser.p12
├── solr
│   ├── ssl-repo-client.keystore
│   └── ssl-repo-client.truststore
└── zeppelin
    ├── ssl-repo-client.keystore
    └── ssl-repo-client.truststore
```

Connecting to SOLR Admin Web Console (by default available in https://127.0.0.1:8983/solr) requires a client certificate

- This certificate needs to be installed in Windows, Mac OS X and Linux.
- When using Mozilla Firefox, the certificate needs also to be installed in that browser.

CLASSIC

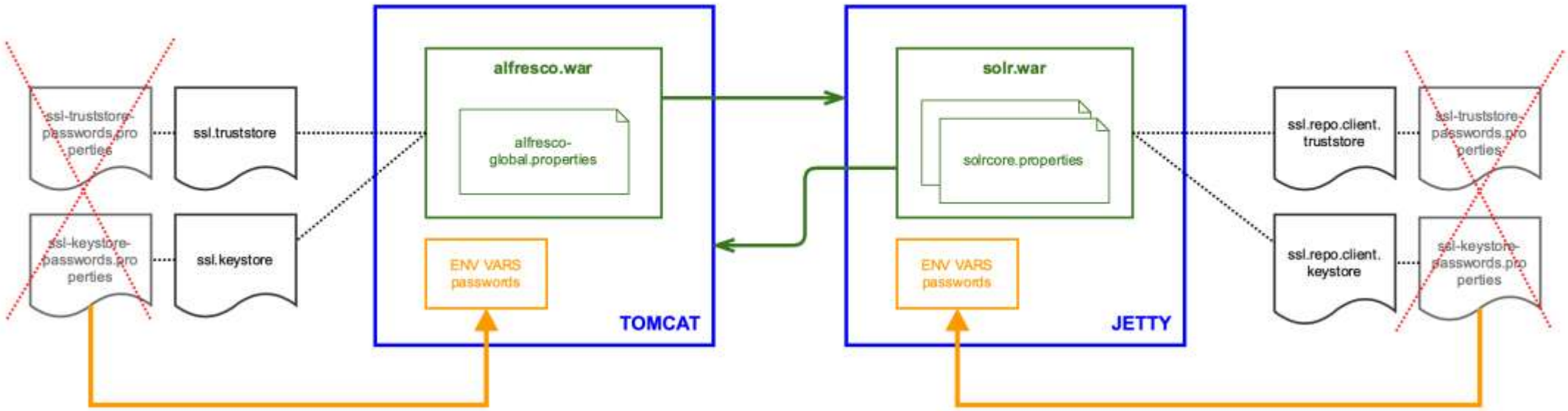
Alfresco mTLS



https://hub.alfresco.com/t5/alfresco-content-services-blog/alfresco-mtls-configuration-deep-dive/ba-p/296422

CURRENT

Alfresco mTLS



CLASSIC

Alfresco mTLS: Repository Properties

alfresco-global.properties
docker-compose.yml

Apache HTTP Client in **alfresco.war** configuration to send HTTPs queries to SOLR

```
# default keystores location
dir.keystore=classpath:alfresco/keystore

# general encryption parameters (keystore)
encryption.keySpec.class=org.alfresco.encryption.DESEDEKeyGenerator
encryption.keyAlgorithm=AES
encryption.cipherAlgorithm=AES/CBC/PKCS5Padding

# secret key keystore configuration
encryption.keystore.location=${dir.keystore}/keystore
encryption.keystore.keyMetaData.location=${dir.keystore}/keystore-passwords.properties
encryption.keystore.provider=
encryption.keystore.type=pkcs12

# ssl.keystore
encryption.ssl.keystore.location=${dir.keystore}/ssl.keystore
encryption.ssl.keystore.provider=
encryption.ssl.keystore.type=JCEKS
encryption.ssl.keystore.keyMetaData.location=${dir.keystore}/ssl-keystore-password.properties

# ssl.truststore
encryption.ssl.truststore.location=${dir.keystore}/ssl.truststore
encryption.ssl.truststore.provider=
encryption.ssl.truststore.type=JCEKS
encryption.ssl.truststore.keyMetaData.location=${dir.keystore}/ssl-truststore-passwords.properties

# SOLR Configuration
solr.port.ssl=8984
solr.secureComms=https
```

ENCRYPTION PROPERTIES
Not related with mTLS Configuration
Required even when not using mTLS

KEYSTORE
Includes Repository private key

TRUSTSTORE
Includes CA public key and
SOLR client public key

https://github.com/Alfresco/alfresco-community-repo/blob/8.307/repository/src/main/resources/alfresco/repository.properties#L719

CLASSIC

Alfresco mTLS: Tomcat Repository Connector

server.xml
Dockerfile

Tomcat Server configuration to receive HTTPs queries from SOLR

TOMCAT CONNECTOR
TLS Configuration

```
$ cat /usr/local/tomcat/conf/server.xml
...
  <Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"
    connectionTimeout="20000" maxThreads="150"

    SSLEnabled="true" scheme="https" secure="true" clientAuth="want" sslProtocol="TLS"

    keystoreFile="/usr/local/tomcat/alf_data/keystore/ssl.keystore"
    keystorePass="kT9X6oe68t" keystoreType="JCEKS"

    truststoreFile="/usr/local/tomcat/alf_data/keystore/ssl.truststore"
    truststorePass="kT9X6oe68t" truststoreType="JCEKS">
  </Connector>
 </Service>
</Server>
```

KEYSTORE
Includes Repository private key

TRUSTSTORE
Includes CA public key and
SOLR client public key

CLASSIC

solrcore.properties

Alfresco mTLS: SOLR Properties

Apache HTTP Client in **solr.war** configuration to send HTTPs indexing requests to Alfresco

```
# ssl.repo.client.keystore
alfresco.encryption.ssl.keystore.type=JCEKS
alfresco.encryption.ssl.keystore.provider=
alfresco.encryption.ssl.keystore.location=ssl.repo.client.keystore
alfresco.encryption.ssl.keystore.passwordFileLocation=ssl-keystore-passwords.properties

# ssl.repo.client.truststore
alfresco.encryption.ssl.truststore.type=JCEKS
alfresco.encryption.ssl.truststore.provider=
alfresco.encryption.ssl.truststore.location=ssl.repo.client.truststore
alfresco.encryption.ssl.truststore.passwordFileLocation=ssl-truststore-passwords.properties

# Alfresco Repository configuration
alfresco.port.ssl=8443
alfresco.secureComms=https
```

KEYSTORE
Includes SOLR private key

TRUSTSTORE
Includes CA public key,
Repository client public key and
SOLR client public key

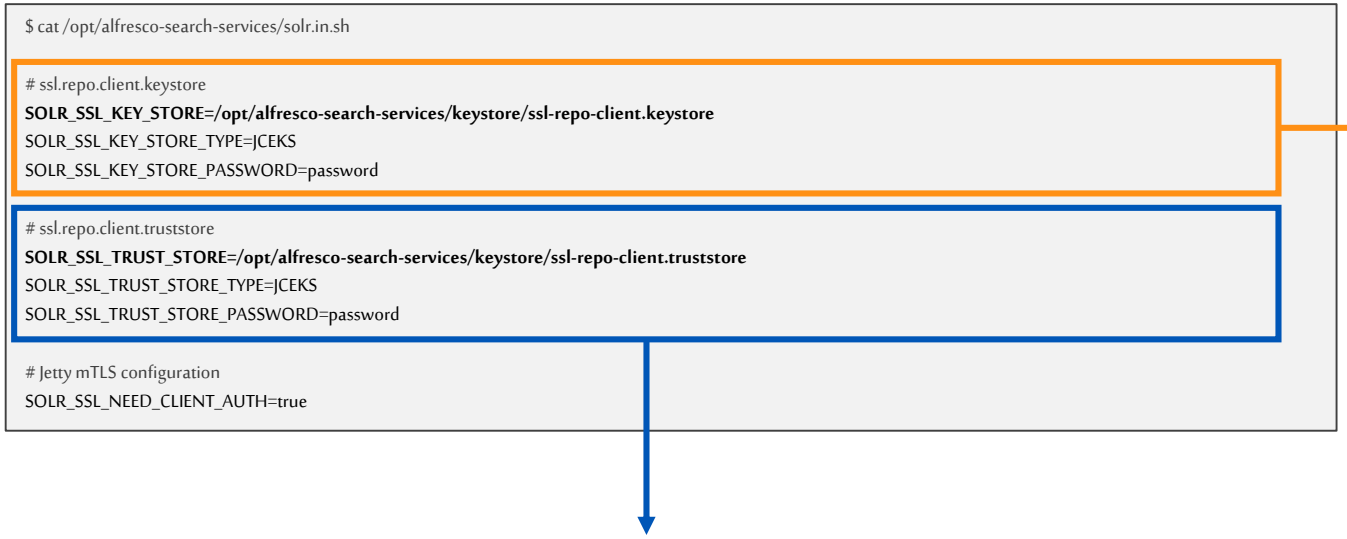
https://github.com/Alfresco/SearchServices/blob/2.0.0/search-services/alfresco-search/src/main/resources/solr/instance/templates/rerank/conf/solrcore.properties#L44

CLASSIC

Alfresco mTLS: Jetty SOLR Server

solr.in.sh
solr.in.cmd

Jetty Server configuration to receive HTTPs queries from Alfresco



KEYSTORE
Includes SOLR private key

TRUSTSTORE
Includes CA public key,
Repository client public key and
SOLR client public key

Alfresco mTLS: SOLR Endpoints

Apache HTTP Client from **alfresco.war** is sending signed HTTPs requests to **SOLR Jetty** server

Search Queries

https://127.0.0.1:8983/solr/alfresco/afts

https://127.0.0.1:8983/solr/alfresco/browse

https://127.0.0.1:8983/solr/alfresco/cmis

https://127.0.0.1:8983/solr/alfresco/query

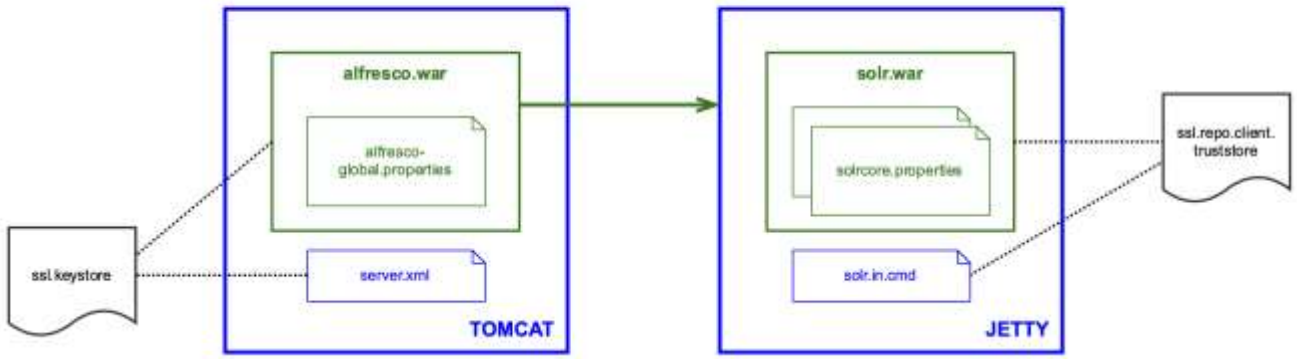
https://127.0.0.1:8983/solr/alfresco/select

SQL Queries

https://127.0.0.1:8983/solr/alfresco/sql

Admin Actions

https://127.0.0.1:8983/solr/admin



Alfresco mTLS: Repository Endpoints

Apache HTTP Client from **solr.war** is sending signed HTTPs requests to **Alfresco Tomcat** server

Indexing requests

https://127.0.0.1:8443/alfresco/service/api/solr/aclchangesets

https://127.0.0.1:8443/alfresco/service/api/solr/acls

https://127.0.0.1:8443/alfresco/service/api/solr/aclsReaders

https://127.0.0.1:8443/alfresco/service/api/solr/metadata

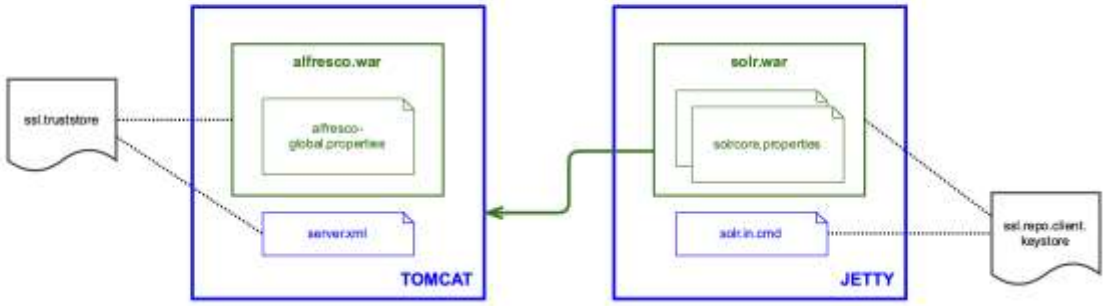
https://127.0.0.1:8443/alfresco/service/api/solr/model

https://127.0.0.1:8443/alfresco/service/api/solr/modelsdiff

https://127.0.0.1:8443/alfresco/service/api/solr/nodes

https://127.0.0.1:8443/alfresco/service/api/solr/textContent

https://127.0.0.1:8443/alfresco/service/api/solr/transactions



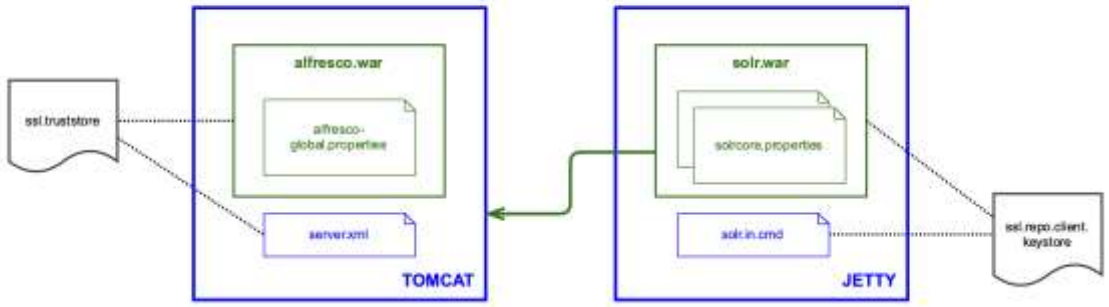
Alfresco mTLS: Sharding

mTLS Configuration can be applied to SOLR Shards in the same way.

- The same KeyStores can be used for every Shard
- A new certificate ssl.client.repo can be generated for each Shard
  - You need to add these new certificates to Alfresco Repository truststore (ssl.truststore)

Sample configuration using DB_ID for two shards is available in:

https://github.com/aborroy/solr-sharding-docker-compose/tree/master/ssl_db_id



CLASSIC

DEMO TIME: Using Custom Certificates

1 - Starting with a working mTLS configuration

- Docker Compose for Alfresco Repository
- ZIP Distribution file for Alfresco Search SOLR

2 - Create new KeyStores with different values

```
$ ./run.sh \
-alfrescoversion community \
-keysize 4096 \
-keystoretype PKCS12 -keystorepass password \
-truststoretype PKCS12 -truststorepass password \
-alfrescoformat classic
```

https://github.com/Alfresco/alfresco-ssl-generator

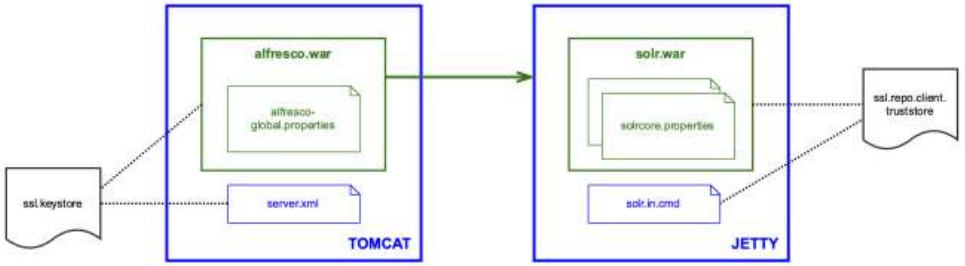
3 - Copy the new KeyStores but preserve encryption resources: keystore and keystore-passwords.properties

4 - Modify configuration in Alfresco Repository, Apache Tomcat, Alfresco Search SOLR and Jetty

- Use pkcs12 as KeyStore Type
- Use password as password for the KeyStores


Panic! at the Disco

Common mistakes: Searching



If you are experimenting problems when **searching**

from Alfresco, Share or from the REST API:

- Review Alfresco *Repository configuration* > alfresco-global.properties

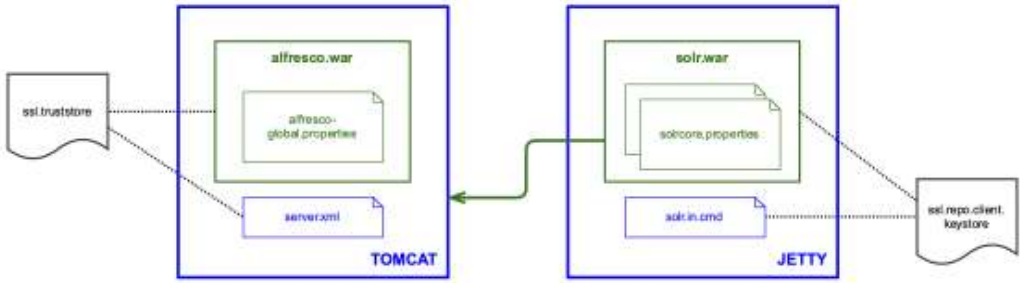
```
solr.port.ssl=8983
solr.secureComms=https

dir.keystore=/usr/local/tomcat/alf_data/keystore

# ssl.keystore
encryption.ssl.keystore.location=${dir.keystore}/ssl.keystore
encryption.ssl.keystore.type=JCEKS
encryption.ssl.keystore.keyMetaData.location=${dir.keystore}/ssl-keystore-password.properties

# ssl.truststore
encryption.ssl.truststore.location=${dir.keystore}/ssl.truststore
encryption.ssl.truststore.type=JCEKS
encryption.ssl.truststore.keyMetaData.location=${dir.keystore}/ssl-truststore-passwords.properties
```

- Review *SOLR Jetty* configuration > solr.in.sh | solr.in.cmd

```
SOLR_SSL_TRUST_STORE=/opt/alfresco-search-services/keystore/ssl.repo.client.truststore
SOLR_SSL_TRUST_STORE_PASSWORD=kT9X6oe68t
SOLR_SSL_TRUST_STORE_TYPE=JCEKS
SOLR_SSL_KEY_STORE=/opt/alfresco-search-services/keystore/ssl.repo.client.keystore
SOLR_SSL_KEY_STORE_PASSWORD=kT9X6oe68t
SOLR_SSL_KEY_STORE_TYPE=JCEKS
SOLR_SSL_NEED_CLIENT_AUTH=true
```

https://hub.alfresco.com/t5/alfresco-content-services-blog/alfresco-mtls-configuration-deep-dive/ba-p/296422

Common mistakes: Indexing



If you are experimenting problems when **indexing** from SOLR:

- Review *Alfresco Tomcat* configuration > server.xml

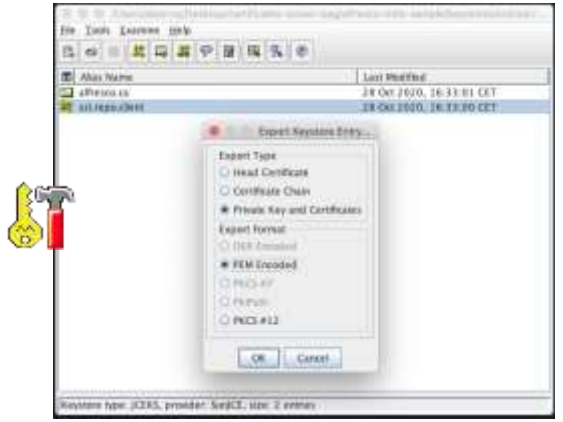
```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"
   connectionTimeout="20000"
   SSLEnabled="true" maxThreads="150" scheme="https"
   keystoreFile="/usr/local/tomcat/alf_data/keystore/ssl.keystore"
   keystorePass="kT9X6oe68t" keystoreType="JCEKS" secure="true"
   truststoreFile="/usr/local/tomcat/alf_data/keystore/ssl.truststore"
   truststorePass="kT9X6oe68t" truststoreType="JCEKS" clientAuth="want" sslProtocol="TLS">
</Connector>
```

- Review *SOLR properties* configuration > solrcore.properties

```
alfresco.secureComms=https
alfresco.port.ssl=8443

alfresco.encryption.ssl.truststore.location=/opt/alfresco-search-services/keystore/ssl.repo.client.truststore
alfresco.encryption.ssl.keystore.provider=JCEKS
alfresco.encryption.ssl.truststore.type=
alfresco.encryption.ssl.keystore.location=/opt/alfresco-search-services/keystore/ssl.repo.client.keystore
alfresco.encryption.ssl.truststore.provider=JCEKS
alfresco.encryption.ssl.truststore.passwordFileLocation=/opt/alfresco-search-services/keystore/ssl-truststore-passwords.properties
alfresco.encryption.ssl.keystore.type=
alfresco.encryption.ssl.keystore.passwordFileLocation=/opt/alfresco-search-services/keystore/ssl-keystore-passwords.properties
```

https://hub.alfresco.com/t5/alfresco-content-services-blog/alfresco-mtls-configuration-deep-dive/ba-p/296422

Troubleshooting: cURL



Testing the configuration with CURL

Extract **ssl.repo.client** certificate from *keystores/solr/ssl.repo.client.keystore* in PEM format:

```
$ curl -k --cert Custom_Alfresco_Repository_Client_Custom_Alfresco_CA.pem –v \
 "https://127.0.0.1:8443/alfresco/service/api/solr/aclchangesets?fromTime=0&toTime=1603454490108&maxResults=2000"
```

In the other way, extract **ssl.repo** certificate from *keystores/alfresco/ssl.keystore* in PEM format

```
$ curl -k --cert Custom_Alfresco_Repository_Custom_Alfresco_CA.pem –v \
 "https://127.0.0.1:8983/solr/alfresco/select?indent=on&q=@sys\:node-dbid:101&wt=json"
```

Troubleshooting: Debugging

Debugging the configuration

The best approach to debug *SSL Handshake* is not using the *Log4j* categories, but setting this Java parameter for both Solr and Alfresco web apps:

-Djavax.net.debug=ssl:handshake

```
"ClientHello": {
 "client version"     : "TLSv1.2",
 "random"             : "79 4D 93 54 F9 59 83 0C 75 58 73 F8 DE 3A 3C B6 95 57 8F 72 A4 FE 92 BB D0 89 50 C3 A0 11 84 9C",
 "session id"         : "49 61 34 4C 45 80 A0 69 75 E3 92 C2 7D F6 2E 04 70 3F 6C 4D A1 91 F0 B8 CE 79 1C 3B 15 0B 11 F5",
 "cipher suites"      : "[TLS_AES_128_GCM_SHA256(0x1301), TLS_AES_256_GCM_SHA384(0x1302), ... ]",
 "compression methods" : "00",
 "extensions"         : [ ]
}
)

"ServerHello": {
 "server version"     : "TLSv1.2",
 "random"             : "30 8C EE E3 E3 08 6D 38 FD BC 47 5E 9A C5 4C A5 AD 14 3E 97 DB 3E DA C9 BE 61 F9 0F 88 F3 25 10",
 "session id"         : "",
 "cipher suite"       : "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384(0xC030)",
 "compression methods" : "00",
 "extensions"         : [
  "renegotiation_info (65,281)": {
   "renegotiated connection": [<no renegotiated connection>]
  },
  "ec_point_formats (11)": {
   "formats": [uncompressed, ansiX962_compressed_prime, ansiX962_compressed_char2]
  }
 ]
}
```

Troubleshooting: Resources

Alfresco Documentation

https://docs.alfresco.com/search-community/tasks/solr-install.html

https://docs.alfresco.com/search-community/concepts/solr-troubleshooting.html

Alfresco Hub

https://hub.alfresco.com/t5/alfresco-content-services-blog/creating-self-signed-ssl-certificates-for-solr/ba-p/288477

https://hub.alfresco.com/t5/alfresco-content-services-blog/using-ssl-with-alfresco-search-services-and-solr-6/ba-p/292687

https://hub.alfresco.com/t5/alfresco-content-services-blog/alfresco-mtls-configuration-deep-dive/ba-p/296422

https://hub.alfresco.com/t5/alfresco-content-services-blog/alfresco-6-1-is-coming-with-mutual-tls-authentication-by-default/ba-p/287905

Blog posts

https://angelborroy.wordpress.com/2016/06/15/configuring-alfresco-ssl-certificates/

Thank you!