NEO4J SSL Setup

TIPS AND BEST PRACTICES

Contents

1. Introduction

This document aims to serve as a best practice guide for setting up SSL for

- Bolt
- HTTPS
- Intra-cluster encryption
- Backup

This document is not a replacement for the SSL framework documentation. The aim of this document is to provide easy steps to set up SSL within your environment and provide best practices. It also provides guidance on viewing, validating and testing the certificates.

The document assumes that the certificates are obtained from the Certificate Authority and will go over using those certificates to set up SSL.

As usual, the Neo4j support team are ready to help you with these operations so feel free to reach out to our support team if you run into any problem or have any questions about the upgrade process (https://support.neo4j.com/hc/en-us).

2. SSL Framework

2.1 Certificates

The SSL support is enabled per communication channel

- bolt (port - 7687)
- https (port - 7473)
- intra-cluster communication (ports - 5000, 6000 and 7000)
- backups (port - 6362)

and requires SSL certificates to be issued by Certificate Authority. All Certificates must be in the PEM format. Private key is also required to be in PEM format.

TIP: *If the same certificates are used across all instances of the cluster, please make sure that when generating the certificates to include the DNS names of all the cluster instances in the certificates. Multi-host and wildcard certificates are supported.*

3. SSL Configuration

The SSL policies are configured by assigning values to parameters of the following format:

```
dbms.ssl.policy.<scope>.<setting-suffix>
```

The scope is the name of the communication channel, and must be one of `bolt`, `https`, `cluster`, `backup` or `fabric`.

The valid values for `setting-suffix` are described below.

Setting suffix	Description	Default value
	`Basic`	
`enabled`	Setting this to `true` will enable this policy.	`false`
`base_directory`	The base directory under which cryptographic objects are searched for by default.	`certificates/<scope>`
`private_key`	The private key used for authenticating and securing this instance.	`private.key`
`private_key_password`	The passphrase to decode the private key. Only applicable for encrypted private keys.	
`public_certificate`	A public certificate matching the private key signed by a Certificate Authority (CA).	`public.crt`
`trusted_dir`	A directory populated with certificates of trusted parties.	`trusted/`
`revoked_dir`	A directory populated with certificate revocation lists (CRLs).	`revoked/`
	`Advanced`	

`verify_hostname`	Enabling this setting will turn on client-side hostname verification. After the client has received the server's public certificate, it will compare the address it used against the certificate Common Name (CN) and Subject Alternative Names (SAN) fields. If the address used doesn’t match those fields, the client will disconnect.	`false`
`ciphers`	A comma-separated list of ciphers suits that will be allowed during cipher negotiation. Valid values depend on the current JRE and SSL provider, see note below for examples.	Java platform default allowed cipher suites
`tls_versions`	A comma-separated list of allowed TLS versions.	`TLSv1.2`
`client_auth`	Whether or not clients must be authenticated. Setting this to REQUIRE effectively enables mutual authentication for servers. Available values given to this setting are NONE, OPTIONAL, or REQUIRE.	`OPTIONAL` for `bolt` and `https` and `REQUIRE` for `cluster` and `backup`.
`trust_all`	Setting this to true will result in all clients and servers being trusted. The content of the trusted_dir directory will be ignored. Use of this is discouraged, since it will not offer security. It is provided as a means of debugging.	`false`

3.1 Configuring SSL over Bolt

Bolt protocol is based on the PackStream serialization and supports the Cypher type system, protocol versioning, authentication and TLS via certificates. For Neo4j Clusters, Bolt provides smart client routing with load balancing and failover. Bolt connector is used by cypher-shell, Neo4j Browser and by the officially supported language drivers.

Bolt connector is enabled as default but its encryption is disabled.

There are 2 main steps to enable encryption over Bolt. First is to create the folder structure and place the key file and the certificates under those. The second step is to make the relevant configuration changes in the neo4j.conf file.

3.1.1 Enable the BOLT Connector

As the very first step to enabling SSL over bolt, is the to enable the following connector

```
dbms.connector.bolt.enabled=true (default is true)
```

3.1.2 Setting up the bolt folder under Certificates

Before setting the neo4j configuration in the neo4j.conf file for enabling the bolt over SSL it is critical to first create the directories in the `NEO4J_HOME/certificates` folder.

- Create a directory `bolt` under `NEO4J_HOME/certificates` folder
- Create a directory `trusted` and `revoked` under `NEO4J_HOME/certificates/bolt` folder

```
$neo4j-home> mkdir certificates/bolt
$neo4j-home> mkdir certificates/bolt/trusted
$neo4j-home> mkdir certificates/bolt/revoked
```

Place the certificates private.key and the public.crt files under the `certificates/bolt` folder.

Place the public.crt file under the `certificates/bolt/trusted` folder.

If a particular `certificate` is revoked then place it under `certificates/bolt/revoked` folder.

The folder structure should look like this with the right file permissions and the groups and ownerships.

Path	Directory /File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/bolt	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/bolt/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/bolt/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/bolt/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/bolt/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/bolt/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

***TIP**: The owner/group should be configured to the user/group that will be running the neo4j service. Default user/group is neo4j/neo4j.*

3.1.3 Setting the bolt SSL configuration in neo4j.conf

There are a couple of configurations that need to be set to enable SSL over bolt once the directory structure and certificates are in place.

Set the SSL Bolt policy to true

```
dbms.ssl.policy.bolt.enabled=true
```

Set the appropriate certificates path and the right key and cert files

```
dbms.ssl.policy.bolt.base_directory=certificates/bolt
dbms.ssl.policy.bolt.private_key=private.key
dbms.ssl.policy.bolt.public_certificate=public.crt
```

***TIP**: If the certificate is a different path outside of NEO4J_HOME then please set the absolute path for the certificates directory.*

Mutual authentication is not enabled for *bolt* hence the below configuration should be set to *NONE*.

```
dbms.ssl.policy.bolt.client_auth=NONE
```

Along with the above configuration, it is recommended to set the following configuration for Bolt as well `dbms.connector.bolt.tls_level`.

It allows the connector to accept encrypted and/or unencrypted connections. The default value is `DISABLED`, where only unencrypted client connections are to be accepted by this connector, and all encrypted connections will be rejected. Other values are `REQUIRED` and `OPTIONAL`. Use `REQUIRED` when only encrypted client connections are to be accepted by this connector, and all unencrypted connections will be rejected. Use `OPTIONAL` where either encrypted or unencrypted client connections are accepted by this connector.

```
dbms.connector.bolt.tls_level=REQUIRED
```

TIP: *In Neo4j version 3.5* `dbms.connector.bolt.tls_level` *is set to default as OPTIONAL. However in the Neo4j 4.x version the default value is DISABLED. Hence please make sure to set this correctly when upgrading from Neo4j version 3.5.x to 4.x.*

3.1.4 Connecting with SSL over Bolt via Drivers

Each of the neo4j and bolt URI schemes permit variants that contain extra encryption and trust information. The +s variants enable encryption with a full certificate check. The +ssc variants enable encryption, but with no certificate check. This latter variant is designed specifically for use with self-signed certificates.

URI Scheme	Routing	Description
`neo4j`	Yes	Unsecured
`neo4j+s`	Yes	Secured with full certificate

`neo4j+ssc`	Yes	Secured with self-signed certificate
`bolt`	No	Unsecured
`bolt+s`	No	Secured with full certificate
`bolt+ssc`	No	Secured with self-signed certificate

To connect to Neo4j once SSL is enabled over Bolt, connection must be made using the neo4j+s or bolt+s.

```
cypher-shell:

cypher-shell -a neo4j+s://<Server DNS or IP>:<Bolt port>
or
cypher-shell -a bolt+s://<Server DNS or IP>:<Bolt port>
```

Example with Java Bolt Driver

```
Java Bolt Driver:

String uri = "neo4j+s://<server dns or ip>:<bolt port>";
Config config = Config.builder()
                     .build();
Driver driver = GraphDatabase.driver(uri, auth, config);

or

String uri = "neo4j://<server dns or ip>:<bolt port>";
Config config = Config.builder()
                     .withEncryption()
                     .build();
Driver driver = GraphDatabase.driver(uri, auth, config);
```

3.2 Configuring SSL over HTTPS

HTTP(s) is used by the Neo4j Browser and the HTTP API. HTTPS (secure HTTP) is set to encrypt network communications.

There are 3 main steps to set SSL over HTTPS. First is to create the folder structure and place the key file and the certificates under those. The second step is to make the relevant configuration changes in the neo4j.conf file.

3.2.1 Enable the HTTPS Connector

As the very first step to enabling SSL over HTTPS, is the to enable the following connector

```
dbms.connector.https.enabled=true (default is false)
```

3.2.2 Setting up the HTTPS folder under Certificates

Before setting the neo4j configuration in the neo4j.conf file for enabling HTTPS over SSL it is critical to first create the directories in the `NEO4J_HOME/certificates` folder.

- Create a directory `https` under `NEO4J_HOME/certificates` folder
- Create a directory `trusted` and `revoked` under `NEO4J_HOME/certificates/https` folder

```
$neo4j-home> mkdir certificates/https
$neo4j-home> mkdir certificates/https/trusted
$neo4j-home> mkdir certificates/https/revoked
```

Place the certificates private.key and the public.crt files under the `certificates/https` folder.

Place the public.crt file under the `certificates/https/trusted` folder.

If a particular `certificate` is revoked then place it under `certificates/https/revoked` folder.

The folder structure should look like this with the right file permissions and the groups and ownerships.

Path	Directory /File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/https	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/https/public.crt	File	neo4j	neo4j	0644	-rw-r--r--

/data/neo4j/certificates/https/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/https/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/https/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/https/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

***TIP**: The owner/group should be configured to the user/group that will be running the neo4j service. Default user/group is neo4j/neo4j.*

3.2.3 Setting the HTTPS ssl configuration in neo4j.conf

There are a couple of configurations that need to be set to enable SSL over https once the directory structure and certificates are in place.

Set the SSL HTTPS policy to true

```
dbms.ssl.policy.https.enabled=true
```

Set the appropriate certificates path and the right key and cert files

```
dbms.ssl.policy.https.base_directory=certificates/https
dbms.ssl.policy.https.private_key=private.key
dbms.ssl.policy.https.public_certificate=public.crt
```

***TIP**: If the certificate is a different path outside of NEO4J_HOME then please set the absolute path for the certificates directory.*

Mutual authentication is not enabled for *https* hence the below configuration should be set to *NONE*.

```
dbms.ssl.policy.https.client_auth=NONE
```

3.2.4 Disable HTTP Connector

Once HTTPS connector with SSL is set up correctly it is recommended from security best practice to disable any communication over HTTP

```
dbms.connector.http.enabled=false
```

3.3 Configuring Intra-Cluster Encryption

Intra-cluster encryption is the security solution for cluster communication that is based on standard SSL/TLS technology (referred to jointly as SSL). The cluster communicates on 3 ports

5000 - Discovery management

6000 - Transactions

7000 - raft communications

There are 3 main steps to set up intra-cluster encryption. First is to create the folder structure and place the key file and the certificates under those. The second step is to make the relevant configuration changes in the neo4j.conf file. The third step is to test that the intra cluster communication is encrypted.

3.3.1 Setting up the folder under Certificates for Intra-Cluster Encryption

Before setting the neo4j configuration in the neo4j.conf file for enabling intra-cluster encryption it is critical to first create the directories in the `NEO4J_HOME/certificates` folder.

- Create a directory `cluster` under `NEO4J_HOME/certificates` folder
- Create a directory `trusted` and `revoked` under `NEO4J_HOME/certificates/cluster` folder

```
$neo4j-home> mkdir certificates/cluster
$neo4j-home> mkdir certificates/cluster/trusted
$neo4j-home> mkdir certificates/cluster/revoked
```

Place the certificates private.key and the public.crt files under the `certificates/cluster` folder.

Place the public.crt file under the `certificates/cluster/trusted` folder.

If a particular `certificate` is revoked then place it under `certificates/cluster/revoked` folder.

The folder structure should look like this with the right file permissions and the groups and ownerships.

Path	Directory /File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/cluster	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/cluster/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/cluster/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/cluster/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/cluster/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/cluster/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

__TIP__: The owner/group should be configured to the user/group that will be running the neo4j service. Default user/group is neo4j/neo4j.

3.3.2 Setting the configuration in neo4j.conf for Intra-Cluster Encryption

There are a couple of configurations that need to be set to enable intra-cluster encryption once the directory structure and certificates are in place.

Set the cluster SSL policy to true

```
dbms.ssl.policy.cluster.enabled=true
```

Set the appropriate certificates path and the right key and cert files

```
dbms.ssl.policy.cluster.base_directory=certificates/cluster
dbms.ssl.policy.cluster.private_key=private.key
dbms.ssl.policy.cluster.public_certificate=public.crt
```

> ***TIP**: If the certificate is a different path outside of NEO4J_HOME then please set the absolute path for the certificates directory.*

For Intra-cluster encryption we deploy a mutual authentication setup, which means that both ends of a channel have to authenticate.

Mutual authentication needs to be enabled for *intra-cluster encryption* hence the below configuration should be set to `REQUIRE`.

```
dbms.ssl.policy.cluster.client_auth=REQUIRE
```

> ***TIP**: If each server has a certificate of its own, signed by a CA, then each server's public certificate has to be put in the trusted folders on each instance of the cluster. Then the server now has the capability of establishing trust with other servers.*

3.3.3 Testing Intra-Cluster Encryption

To make sure that everything is secured as intended it makes sense to validate using external tooling such as nmap.

The command to test is as follows

```
nmap --script ssl-enum-ciphers -p <port> <hostname>
```

The hostname and port have to be adjusted according to our configuration. This can prove that TLS is in fact enabled and that only the intended cipher suites are enabled. All servers and all applicable ports should be tested.

If the intra-cluster encryption is enabled, the output should indicate the port is open and it is using TLS with the ciphers used. .

Examples:

```
Testing intra-clustering encryption on port 7000 (for raft communication)


nmap --script +ssl-enum-ciphers 10.195.12.20 -p 7000
```

```
Starting Nmap 6.40 ( http://nmap.org ) at 2020-01-03 15:06 UTC
Nmap scan report for neo4jservername.dns.com (10.195.12.20)
Host is up (0.000037s latency).
PORT     STATE SERVICE
7000/tcp open  afs3-fileserver
| ssl-enum-ciphers:
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 - strong
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA - strong
|       TLS_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_RSA_WITH_AES_128_GCM_SHA256 - strong
|       TLS_RSA_WITH_AES_256_CBC_SHA - strong
|     compressors:
|       NULL
|_  least strength: strong
```

```
Testing intra-clustering encryption on port 6000 (for transaction communication)

nmap --script +ssl-enum-ciphers 10.195.12.20 -p 6000

Starting Nmap 6.40 ( http://nmap.org ) at 2020-01-03 15:06 UTC
Nmap scan report for neo4jservername.dns.com (10.195.12.20)
Host is up (0.000031s latency).
PORT     STATE SERVICE
6000/tcp open  X11
| ssl-enum-ciphers:
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 - strong
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA - strong
|       TLS_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_RSA_WITH_AES_128_GCM_SHA256 - strong
|       TLS_RSA_WITH_AES_256_CBC_SHA - strong
|     compressors:
|       NULL
|_  least strength: strong
```

3.4 Configuring SSL For Backup Communications

In a single instance by default backup communication happens on port 6362. In a cluster topology, it is possible to take a backup from any server, and each server has two configurable ports capable of serving a backup. These ports are configured by `dbms.backup.listen.address (port 6362)` and `causal_clustering.transaction_listen_address (port 6000)` respectively. If the intra-cluster encryption is already enabled and if backup communication is using port 6000 then the below setup is not required.

However the below steps for securing backup communication are required only if backup is set up on a different port.

There are 2 main steps to set SSL for backup communication. First is to create the folder structure and place the key file and the certificates under those. The second step is to make the relevant configuration changes in the neo4j.conf file.

3.4.1 Setting up the BACKUP folder under Certificates

Before setting the neo4j configuration in the neo4j.conf file for enabling SSL for backups communication it is critical to first create the directories in the `NEO4J_HOME/certificates` folder.

- Create a directory `backup` under `NEO4J_HOME/certificates` folder
- Create a directory `trusted` and `revoked` under `NEO4J_HOME/certificates/https` folder

```
$neo4j-home> mkdir certificates/backup
$neo4j-home> mkdir certificates/backup/trusted
$neo4j-home> mkdir certificates/backup/revoked
```

Place the certificates private.key and the public.crt files under the `certificates/backup` folder.

Place the public.crt file under the `certificates/backup/trusted` folder.

If a particular `certificate` is revoked then place it under `certificates/backup/revoked` folder.

The folder structure should look like this with the right file permissions and the groups and ownerships.

Path	Directory /File	Owner	Group	Permission	Unix/Linux View
/data/neo4j/certificates/backup	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/backup/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/backup/private.key	File	neo4j	neo4j	0400	-r--------
/data/neo4j/certificates/backup/trusted	Directory	neo4j	neo4j	0755	drwxr-xr-x
/data/neo4j/certificates/backup/trusted/public.crt	File	neo4j	neo4j	0644	-rw-r--r--
/data/neo4j/certificates/backup/revoked	Directory	neo4j	neo4j	0755	drwxr-xr-x

TIP: The owner/group should be configured to the user/group that will be running the neo4j service. Default user/group is neo4j/neo4j.

3.4.2 Setting the BACKUP ssl configuration in neo4j.conf

There are a couple of configurations that need to be set to enable SSL for backup communication once the directory structure and certificates are in place.

Set the SSL BACKUP policy to true

```
dbms.ssl.policy.backup.enabled=true
```

Set the appropriate certificates path and the right key and cert files

```
dbms.ssl.policy.backup.base_directory=certificates/backup
dbms.ssl.policy.backup.private_key=private.key
dbms.ssl.policy.backup.public_certificate=public.crt
```

TIP: If the certificate is a different path outside of NEO4J_HOME then please set the absolute path for the certificates directory.

Mutual authentication is not enabled for *backup* hence the below configuration should be set to *REQUIRE*.

```
dbms.ssl.policy.backup.client_auth=REQUIRE
```

3.5 Other Configurations for SSL

3.5.1 Using Encrypted Private Key

If the private key is encrypted, then the following configuration is required

```
Bolt:

dbms.ssl.policy.bolt.private_key_password=<clear text password>
```

```
https:

dbms.ssl.policy.https.private_key_password=<password>
```

```
Intra-cluster encryption:

dbms.ssl.policy.cluster.private_key_password=<password>
```

```
backup:

dbms.ssl.policy.backup.private_key_password=<password>
```

The private key password should be clear text format without any quotes. If hardcoding of clear text private key password is not feasible due to security constraints, it can be set up to use dynamic password pickup

```
        dbms.ssl.policy.bolt.private_key_password=$(openssl aes-256-cbc -a -d -in
/opt/neo4j/etc/neo4j/neo4j_cert_pwd.enc -kfile
/opt/neo4j/var/lib/neo4j/certificates/bolt/neo4j.cert)
```

If using the above dynamic command, the neo4j should be started using the command expansion option as given in the documentation.

3.5.2 Using Specific Cipher for SSL Configuration

There are cases where Enterprise requires the use of specific ciphers for encryptions. One can set up a neo4j configuration by specifying the list of cipher suits that will be allowed during cipher negotiation. Valid values depend on the current JRE and SSL provider.

For Oracle JRE here is the list of supported ones - https://docs.oracle.com/en/java/javase/11/docs/specs/security/standard-names.html#jsse-cipher-suite-names

```
Bolt:

dbms.ssl.policy.bolt.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

```
https:

dbms.ssl.policy.https.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

```
Intra-cluster encryption:

dbms.ssl.policy.cluster.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

```
backup:

dbms.ssl.policy.backup.ciphers=TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

3.5.3 OCSP Stapling

Starting with Neo4j version 4.2, Neo4j has implemented OCSP Stapling. This can be configured via the neo4j configuration file. OCSP Stapling is implemented on the server side and also available for Java Bolt driver and HTTP API. It is currently not available across other language drivers (Javascript, Python, DotNet and Go).

3.5.3.1 Configuring OCSP Stapling

OCSP Stapling can be enabled on the server side in the neo4j.conf file using the following configuration `dbms.connector.bolt.ocsp_stapling_enabled (default = false).`

```
dbms.connector.bolt.ocsp_stapling_enabled=true
```

Important point is to make sure that SSL over bolt is enabled `(dbms.ssl.policy.bolt.enabled=true).`

3.5.3.2 OCSP Stapling - Driver Implementation

As stated OCSP Stapling is supported for Bolt and HTTP API. Driver implementation for OCSP Stapling is only available for Java language drivers. For driver side configuration we give the users a choice of 3 options:

1) Revocation Checking Disabled (default)

2) Revocation Checking If Present (so-called "Soft fail") - The client will request stapled responses via the "status_request" extension in the TLS handshake but if the server does not return any stapled responses then the client will proceed with certificate validation.

3) Revocation Checking enforced (so-called "Hard Fail") - The client requests stapled responses via the "status_request" extensions in the TLS handshake and the client will fail validation if the server does not return any stapled responses.

The drivers diverge on how trust settings are configured and therefore configuration will vary. As an illustrative example, the java driver will be configurable with methods on `TrustStrategy` as follows:

```
withoutRevocationChecks()
or
withVerifyIfPresentRevocationChecks()
or
withStrictRevocationChecks()
```

Example of the Java driver code:

```
Config config = Config.builder()
      .withMaxConnectionLifetime(30, TimeUnit.MINUTES)

      .withMaxConnectionPoolSize(50)

      .withEncryption()

      .withTrustStrategy( Config.TrustStrategy

              .trustSystemCertificates()

              .withVerifyIfPresentRevocationChecks() )
      .withConnectionAcquisitionTimeout(2, TimeUnit.MINUTES)
      .build();

String uri = "neo4j://single.neo4jcustomersuccess.com:7687";
String user = "neo4j";
String password = "XXXXXXXX";

Driver driver = GraphDatabase.driver(uri, AuthTokens.basic(user, password),
config);
```

4. Troubleshooting

4.1 Viewing, Validating and Transforming Certificates and Keys

4.1.1 Testing the Bolt and HTTPS over SSL

Test the SSL connection to the specified host and bolt/https port and view the certificate.

```
openssl s_client -connect my_domain.com:7687  #bolt
openssl s_client -connect my_domain.com:7473  #https
```

4.1.2 Validating the Key and the Certificate

Validate the key file and the certificate as follows

```
#validate the key
openssl rsa -in private.key -check

#validate certificate in the PEM format
PEM - $openssl x509 -in public.crt –text –noout
DER - $openssl x509 -in certificate.der -inform der -text -noout   (<--- only
use PEM formatted certificates/keys)
```

4.1.3 Transforming the Certificates

Neo4j requires all SSL certificates to be in the PEM format. So it is essential that if the certificate is obtained in the DER format that it is transformed into PEM format.

Also one might need to transform the certificate from PEM to DER format

```
#Transform DER format certificate to PEM format
openssl x509 -in cert.crt -inform der -outform pem -out cert.pem
```

```
#Transform PEM format certificate to DER format
openssl x509 -in cert.crt -outform der -out cert.der
```

4.2 Server Side debugging

Neo4j provides various log files for monitoring purposes. Debug.log provides information if there are any errors related to the SSL.

4.2.1 Enable Additional Debug Logging

Additional debugging can be enabled by adding the following configuration in the neo4j.conf file

```
dbms.jvm.additional=-Djavax.net.debug=ssl:handshake
```

After enabling this configuration, stop and start the neo4j service. This will log additional information in neo4j.log. In some installations done using rpm based installs neo4j.log is not created. To get the contents of this, since neo4j.log just contains STDOUT content, simply look for the neo4j service log contents using `journalctl`:

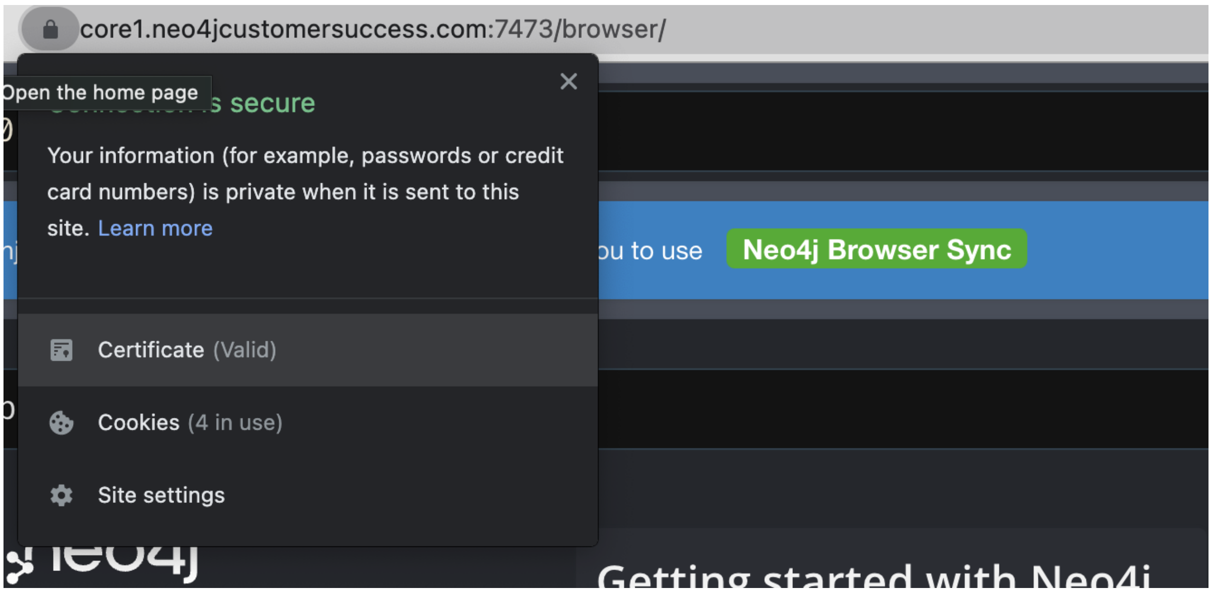
```
neo4j@ubuntu:/var/log/neo4j$ journalctl -u neo4j -b > neo4j.log
neo4j@ubuntu:/var/log/neo4j$ vi neo4j.log
```

4.2.2 Certificate Unknown Error

If one sees the following error `Received fatal alert: certificate_unknown` in the debug.log

```
2020-05-05 16:36:37.034+0000 ERROR [o.n.b.t.TransportSelectionHandler] Fatal
error occurred when initialising pipeline: [id: 0x000ae28a, L:/127.0.0.1:7687 !
R:/127.0.0.1:36930] javax.net.ssl.SSLHandshakeException: Received fatal alert:
certificate_unknown
```

This error generally indicates that the client (browser) does not have a valid certificate. This can be verified by clicking on the lock icon in the Browser URI.


core1.neo4jcustomersuccess.com:7473/browser/
Open the home page
Connection is secure
Your information (for example, passwords or credit card numbers) is private when it is sent to this site. Learn more
Certificate (Valid)
Cookies (4 in use)
Site settings
Neo4j Browser Sync
Getting started with Neo4j

So if this error is encountered this would say `Certificate (Invalid)`. In the example above the certificate is valid.

5. Appendix

5.1 References

Neo4j SSL Configuration Best Practices

Blog Post - Getting Certificates for Neo4j with LetsEncrypt

Neo4j Documentation - SSL Framework

Neo4j Documentation - Intra-Cluster Encryption