Enable TLS on SQL Server

https://4sysops.com/archives/enable-tls-on-sql-server/

Secure Socket Layer (SSL) and Transport Layer Security (TLS, which builds on the now deprecated SSL protocol) allow you to encrypt Microsoft SQL Server network communication, which is essential for your security. In this post, you'll learn how to enable TLS on your SQL Server.

Contents

Before jumping directly into the configuration, let me first briefly cover some background information. As an enterprise-class database product, SQL Server supports encryption-at-rest using transparent data encryption (TDE), but in this post, we will focus on encryption-in-transit using an SSL/TLS certificate.

SQL Server has long supported SSL encryption, but due to various vulnerabilities in the earlier versions of the SSL protocol (e.g., SSLv2, SSLv3, and TLSv1.0), Microsoft eventually stopped supporting SSL and announced the support of TLS 1.2 back in January 2016. Since then, Microsoft has recommended using TLS 1.2 for encryption with all versions of SQL Server (including older versions, such as 2008, 2008R2, 2012, and 2014).

To enable encryption, we will install a Transport Layer Security (TLS) certificate on SQL Server, configure the SQL Server instance to use this certificate, and enforce encryption so that communication between the SQL Server and the client application is always secured. Once encryption is forced, beware that SQL Server will start rejecting client connections that do not support encryption. Thus, it is important that you first try these steps in a development or staging environment to avoid downtime of your production system.

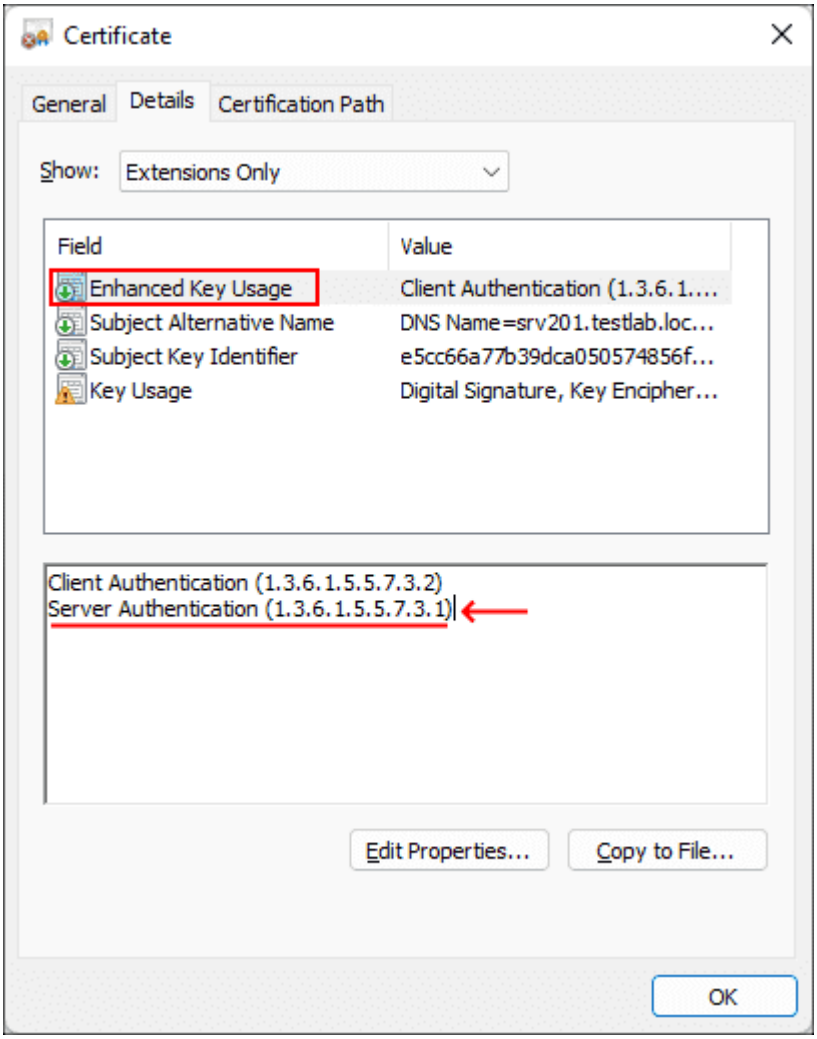
Installing the TLS certificate

Purchase a certificate from a trusted CA

You can obtain a certificate from any trusted public certificate authority (such as DigiCert, Comodo, or GoDaddy). After purchasing the certificate, you need to submit a certificate signing request (CSR), issue the certificate, and install it on your SQL Server. The procedure might vary according to the certificate provider.

The TLS certificate requires the following attributes:

- The object identifier (OID) under enhanced key usage should be *1.3.6.1.5.5.7.3.1.* OID is a numeric value used to identify the purpose of a certificate. The certificate authority automatically includes OID in the enhanced key usage field while creating a certificate for you. The OID *1.3.6.1.5.5.7.3.1* essentially indicates that the certificate could be used for server authentication.



Viewing the enhanced key usage extension of a TLS certificate

The following table shows some of the most commonly used OIDs, along with their purpose:

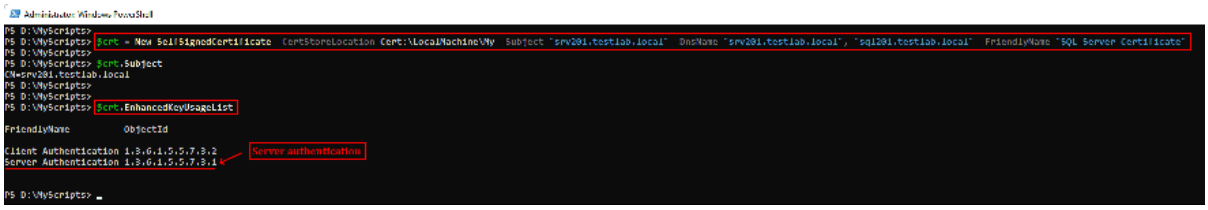
OID	Purpose
1.3.6.1.5.5.7.3.1	Server authentication
1.3.6.1.5.5.7.3.2	Client authentication
1.3.6.1.5.5.7.3.3	Code signing
1.3.6.1.5.5.7.3.4	Email protection
1.3.6.1.5.5.7.3.8	Time stamping
1.3.6.1.5.5.7.3.9	OSCP signing

- The common name (CN) should match the host name or fully qualified domain name (FQDN) of the SQL Server.
- If you address the SQL Server instance with any alternate name, the Subject Alternate Name (SAN) field should also include that name.

Generate a self-signed certificate with PowerShell

For demonstration purposes, we will use PowerShell to generate a self-signed certificate. To do so, launch an elevated PowerShell console on the SQL Server host, and run the following command:

```
New-SelfSignedCertificate -CertStoreLocation Cert:\LocalMachine\My -Subject "srv201.testlab.local" -DnsName "srv201.testlab.local", "sql201.testlab.local" -FriendlyName "SQL Server Certificate"
```



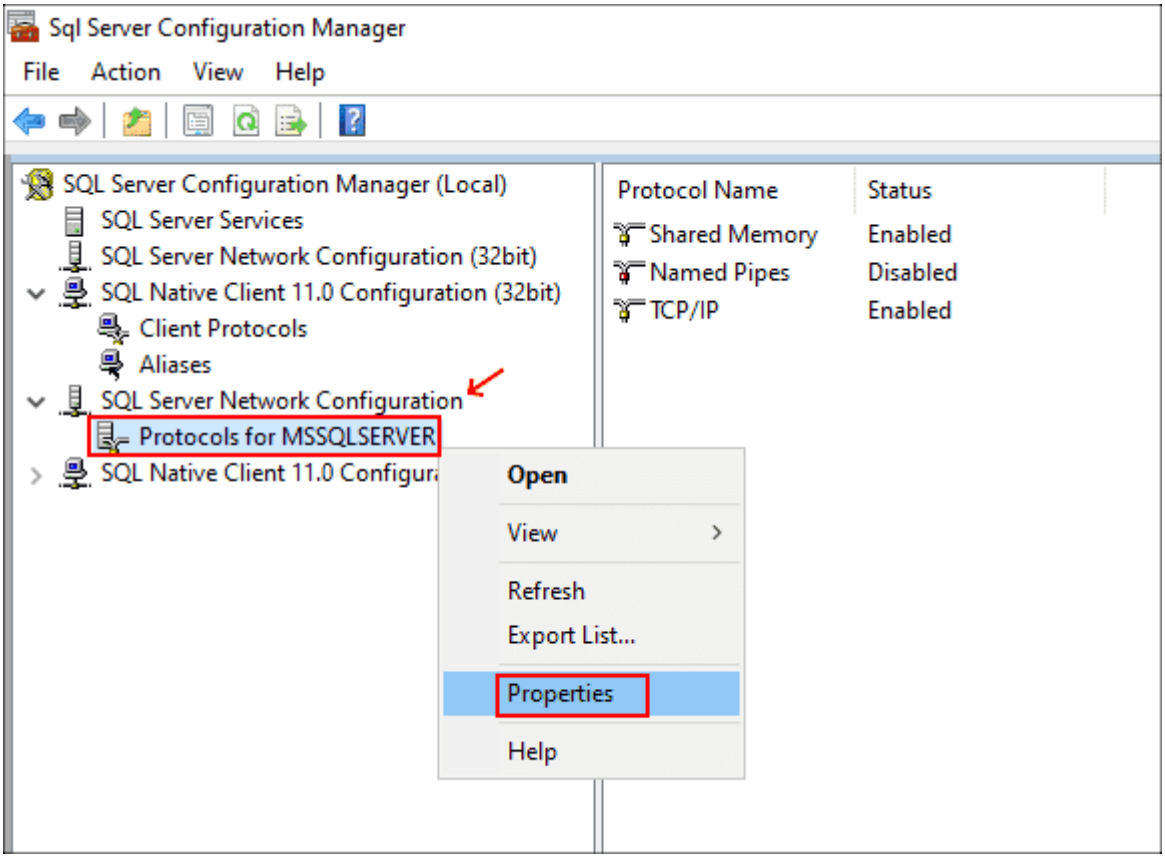
Generating a self signed certificate using PowerShell

Make sure you specify the right name to match your SQL Server with the *-Subject* parameter. The optional *-DnsName* parameter specifies a comma-separated list of subject alternative names (SANs). This command generates a self-signed certificate and saves it in the local computer store. Beware—the self-signed certificates are susceptible to man-in-the-middle (MitM) attacks, so it is highly recommended to obtain a valid certificate from a trusted public certificate authority if you're setting up encryption on a production SQL Server.

Bind certificate to SQL Server instance

Once you have the certificate installed, you need to bind it to the SQL Server instance. To do so, follow these steps:

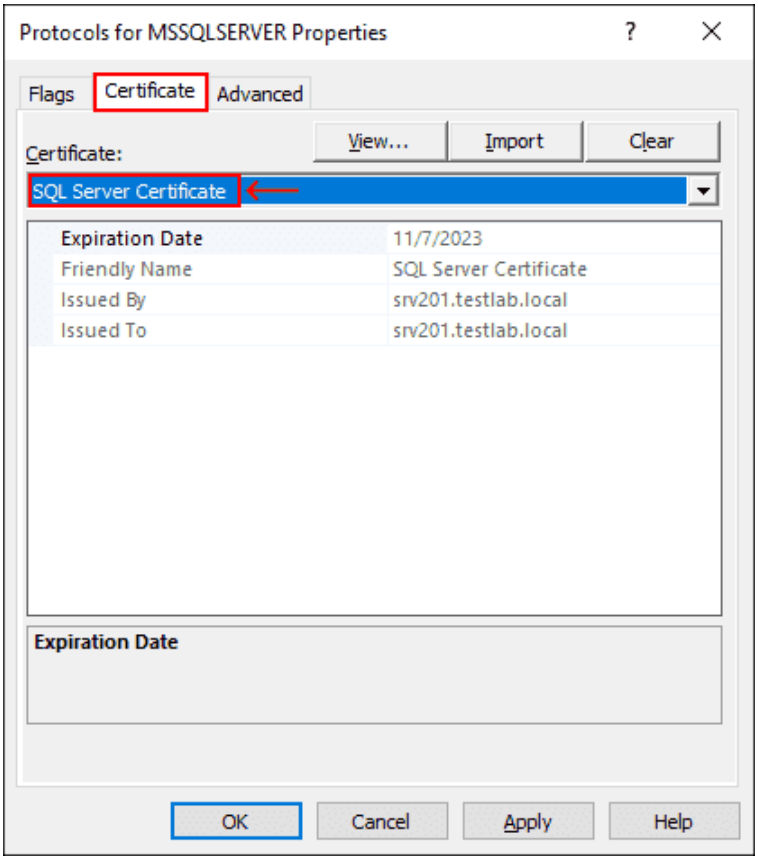
1. Open SQL Server Configuration Manager.
2. Click *SQL Server Network Configuration*, right-click *Protocols for MSSQLSERVER*, and select *Properties* from the context menu, as shown in the following screenshot:



Opening the properties of the SQL Server instance in SQL Server Configuration Manager

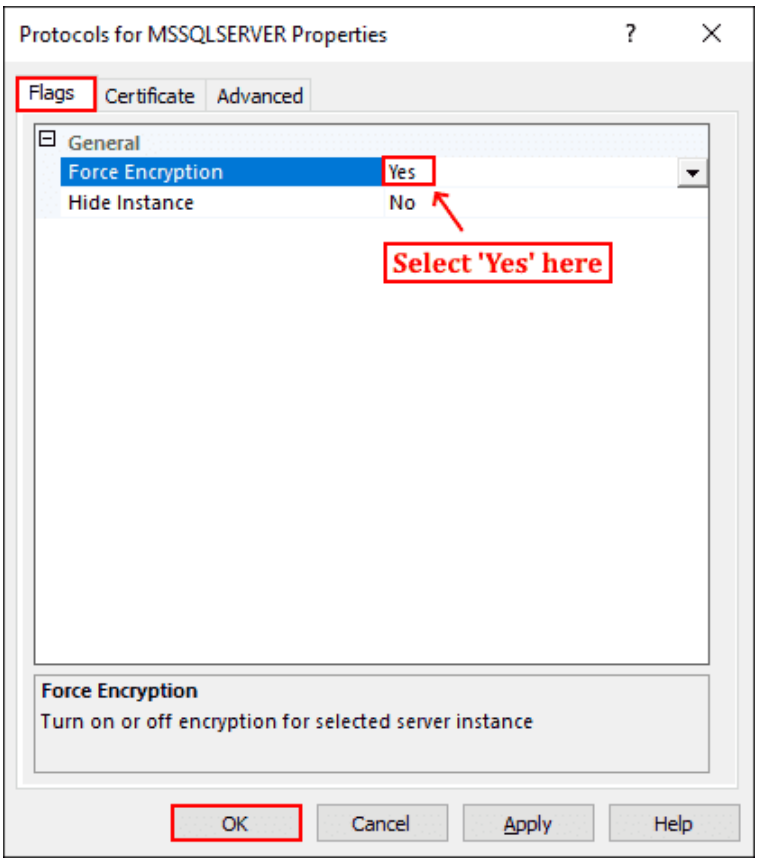
This opens the Protocols for MSSQLSERVER Properties dialog box.

3. Click the *Certificate* tab, and choose the TLS certificate you installed from the dropdown list.



Binding the TLS certificate with the SQL Server instance using SQL Server Configuration Manager

4. Now click the *Flags* tab, select Yes under the *Force Encryption* field, and click OK. If you skip this step, the SQL Server will allow both encrypted and unencrypted connections from clients; however, doing so is not recommended, as it defeats the purpose of this guide.



Force encryption in the SQL Server instance using SQL Server Configuration Manager

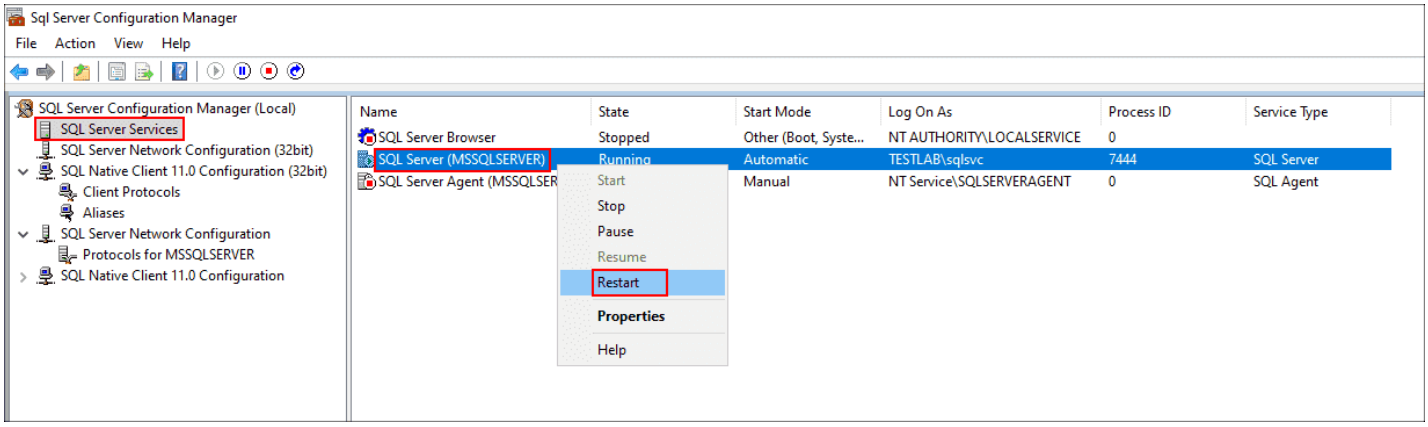
You are shown a dialog box saying *Any changes made will be saved; however, they will not take effect until the service is stopped and restarted*.

5. Click OK to confirm.

Any changes made will be saved however they will not take effect until the service is stopped and restarted

Any changes made will be saved; however, they will not take effect until the service is stopped and restarted

6. Finally, click the *SQL Server Services* node, right-click *SQL Server (MSSQLSERVER),* and select *Restart*



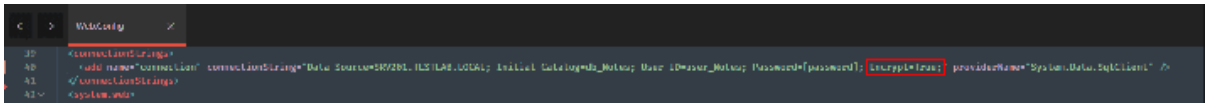
Restarting the SQL Server database service using SQL Server Configuration Manager

from the context menu. This restarts the SQL Server database service.

If you're using SQL Server in a clustered environment, you need to follow these steps on all SQL Server nodes. You also need to make sure that the SAN field includes the virtual network name of the SQL Server instance. Your SQL Server is now ready to accept encrypted connections.

Enable TLS on client applications ^

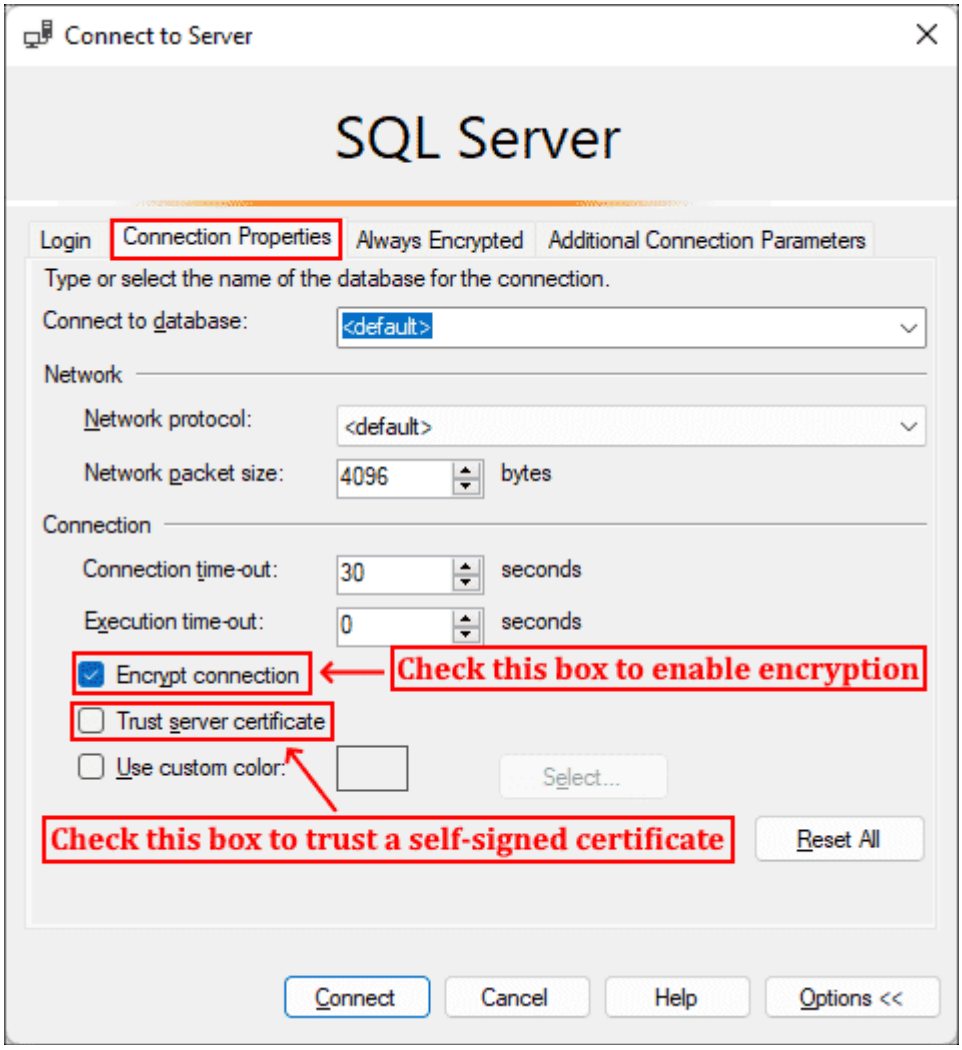
Now that you have configured your SQL Server instance to force encryption, it will accept only encrypted connections. However, you also need to configure your client applications to support the encrypted connection. To do so, modify the connection string in the application's web.config file to include *Encrypt=True*, as shown in the screenshot below:



Modify the connection string in the web.config file to support SQL Server connection encryption

To get additional help with this, speak to your application developer.

In SQL Server Management Studio (SSMS), you can modify the *Connection Properties*, as shown in the following screenshot:



Enable connection encryption in SQL Server Management Studio

Let's discuss some errors that you may encounter and how to fix them.

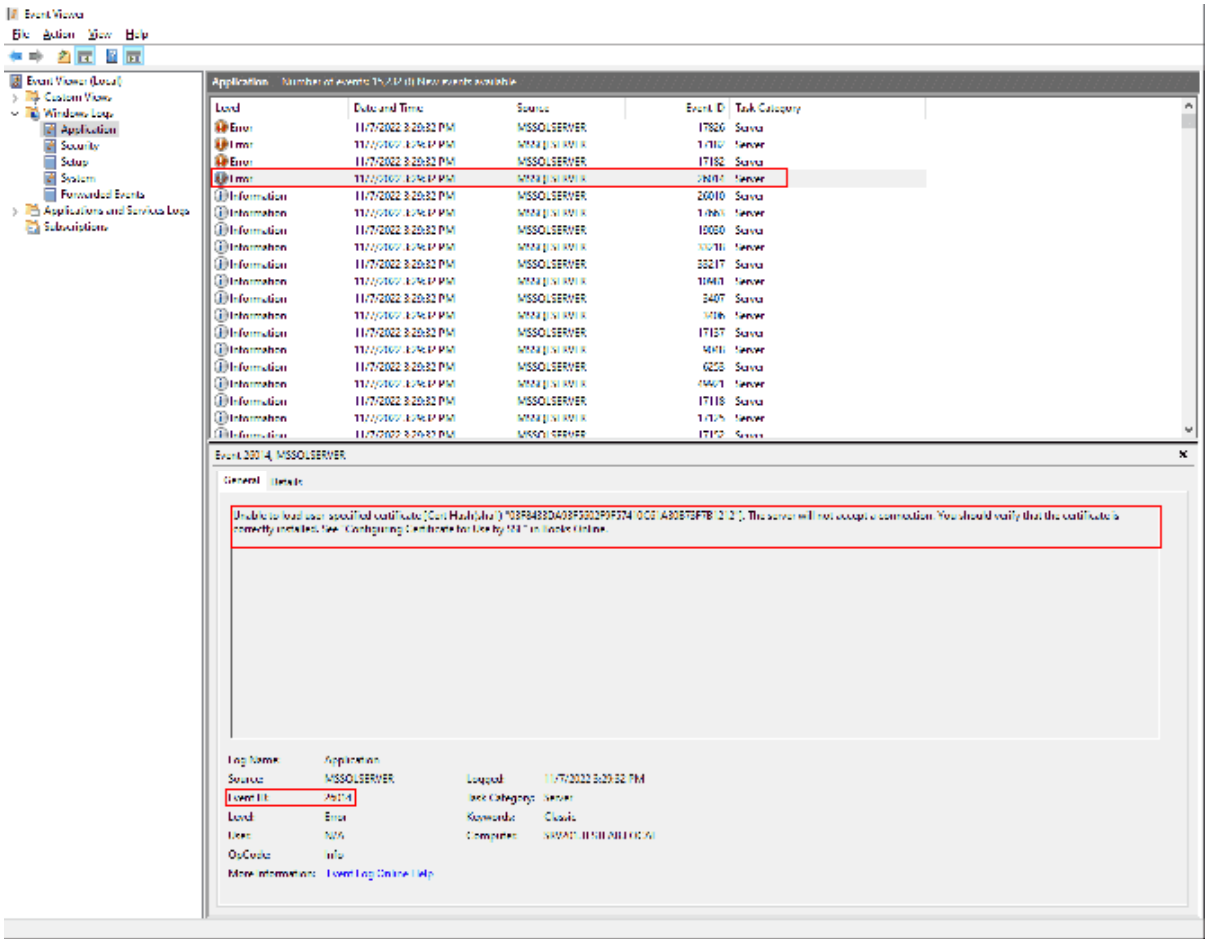
Errors when enabling TLS on SQL Server ^

SQL Server service not starting

If your SQL Server database service won't start after following the above steps, the SQL Server service is most likely running under a custom (managed) service account. In that case, the service account won't be able to load the certificate due to insufficient permissions on the private key, and you will see this error message in the Event Viewer:

Unable to load user-specified certificate [Cert Hash(sha1) "03F8433DA93F5602F9F57410C61A30B73F7B1212"]. The server will not accept a connection. You should verify that the certificate is correctly installed.

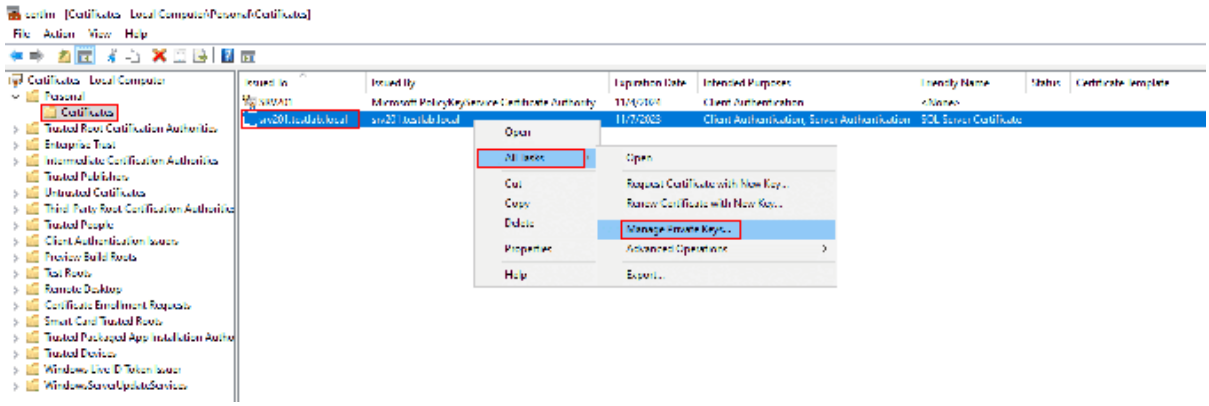
The corresponding event ID in the Event Viewer is 26014.



Event ID 26014 Unable to load user specified certificate. The server will not accept a connection.

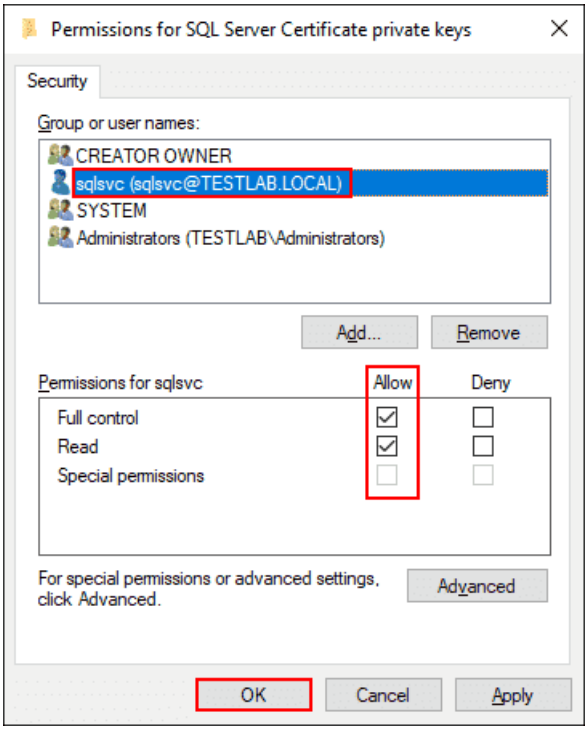
To fix this error, follow these steps:

1. Open the Run dialog box, type *certlm.msc* (or *cermgr.msc)* to open up the certificate manager for the local computer or the current user (depending upon where you installed the TLS certificate), and press Enter.
2. Now, locate the TLS certificate that is bound to the SQL Server instance. Right-click the certificate, select *All Tasks,* and select *Manage Private Keys*, as shown in the screenshot below:



Certificate manager for the local computer Manage private keys

3. In the permissions dialog box, add the SQL Server service account, grant *Full control* permission, and click OK.



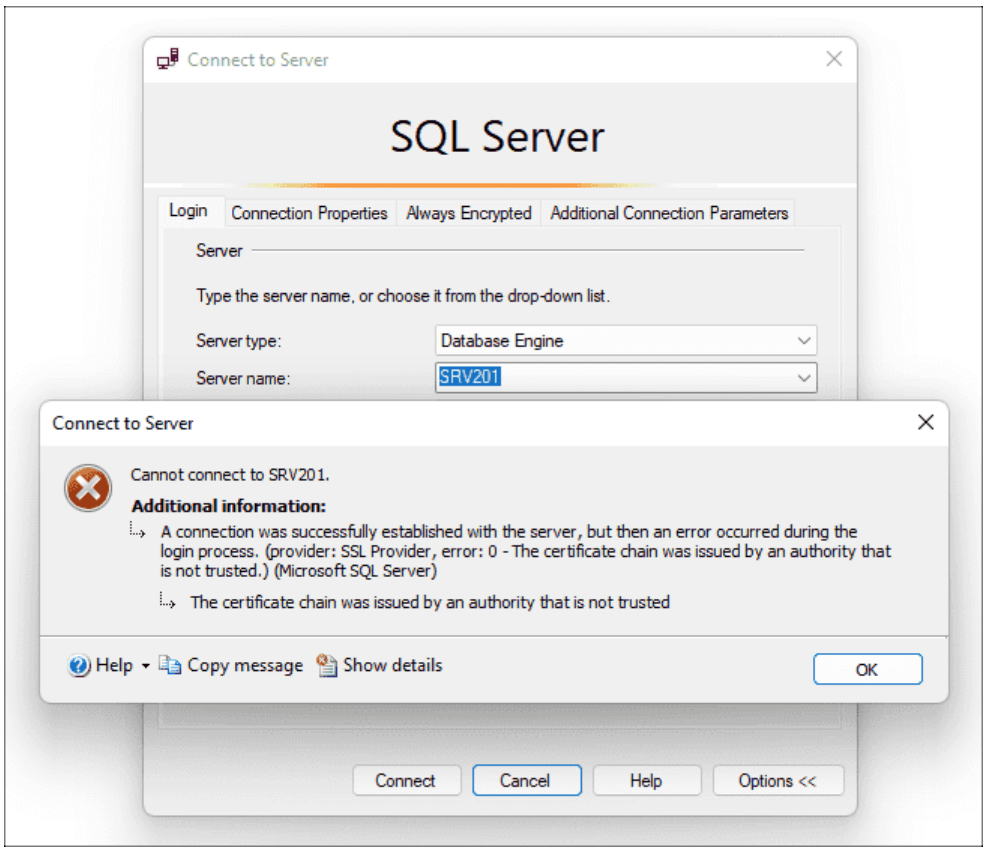
Grant full control permission to the SQL Server service account on the certificate private key

4. Try restarting the SQL Server service again, and ensure it will start without any error.

Self-signed certificate: Error 2146893019

If you used a self-signed certificate with SQL Server, you will likely see the following error in SSMS:

A connection was successfully established with the server, but then an error occurred during the login process. (provider: SSL Provider, error: 0 - The certificate chain was issued by an authority that is not trusted.) (Microsoft SQL Server, Error: -2146893019)



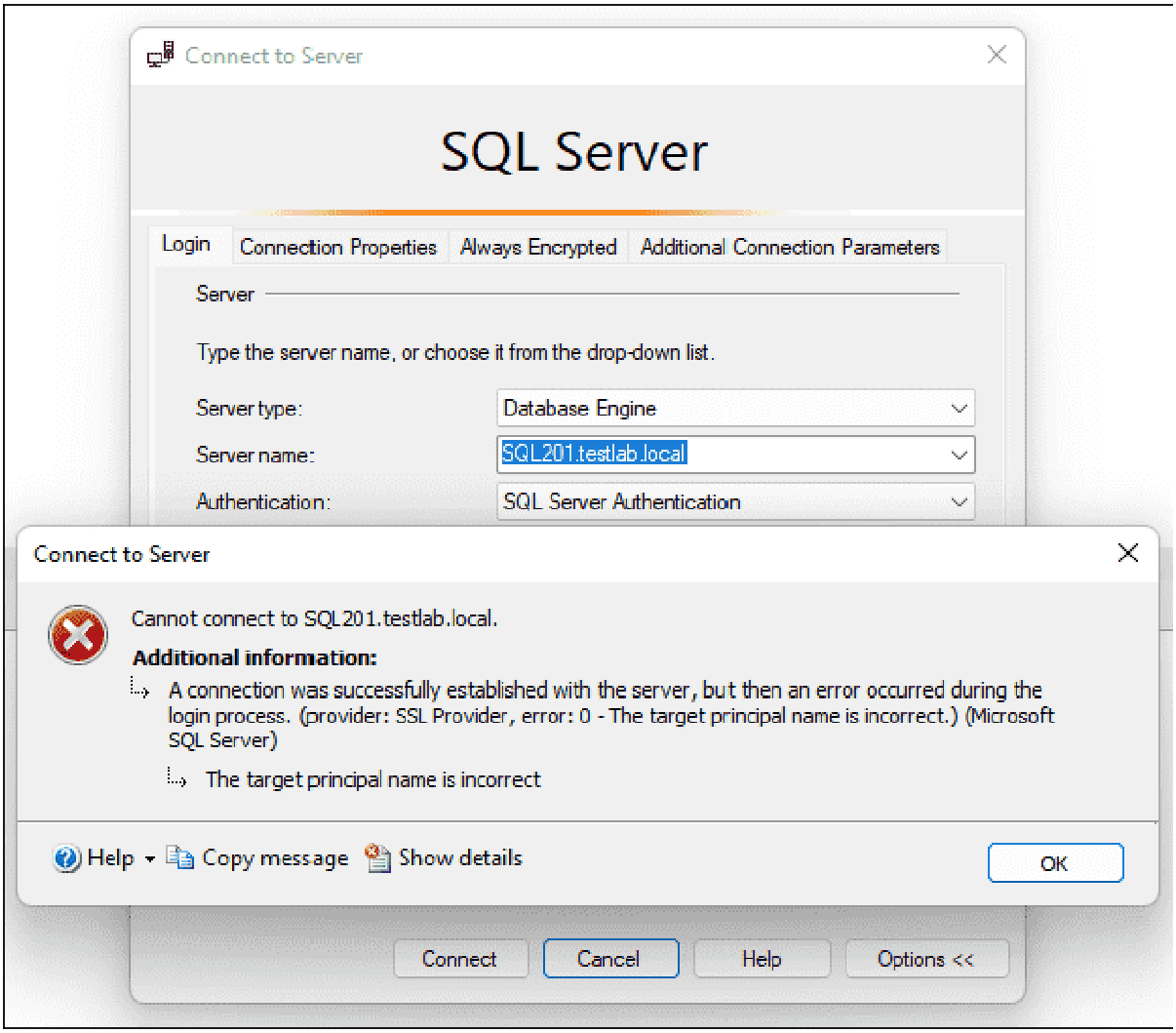
The certificate chain was issued by an authority that is not trusted. Microsoft SQL Server error 2146893019

To fix this error, you can select the *Trust server certificate* option in *Connection Properties* in SSMS. Again, it is highly recommended to use a valid certificate from a trusted certificate authority to avoid such errors and minimize the potential risks of man-in-the-middle (MitM) attacks.

Target principal name is incorrect: Error 2146893022

You might also receive the following error:

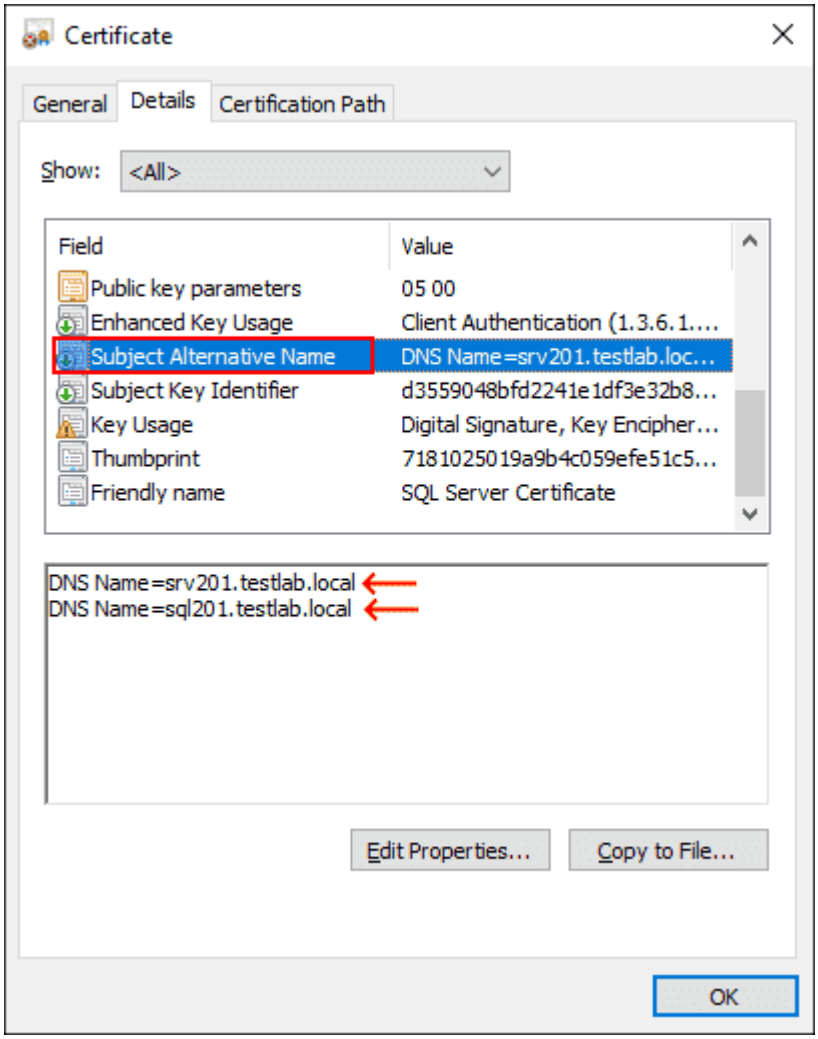
A connection was successfully established with the server, but then an error occurred during the login process. (provider: SSL Provider, error: 0 - The target principal name is incorrect.) (Microsoft SQL Server, Error: -2146893022)



The target principals name is incorrect. Microsoft SQL Server error 2146893022

The error occurs because the name of the SQL Server that you're trying to connect to doesn't match the common name (CN) and also isn't available in the list of *subject alternate names* (SANs).

To fix this error, you need to regenerate the TLS certificate and include the principal name under DNS Name in the SAN. If you used a PowerShell command to generate the certificate, use the *-DnsName* parameter to add all the alternative names. The SAN field should list all the DNS names, as shown in the screenshot:



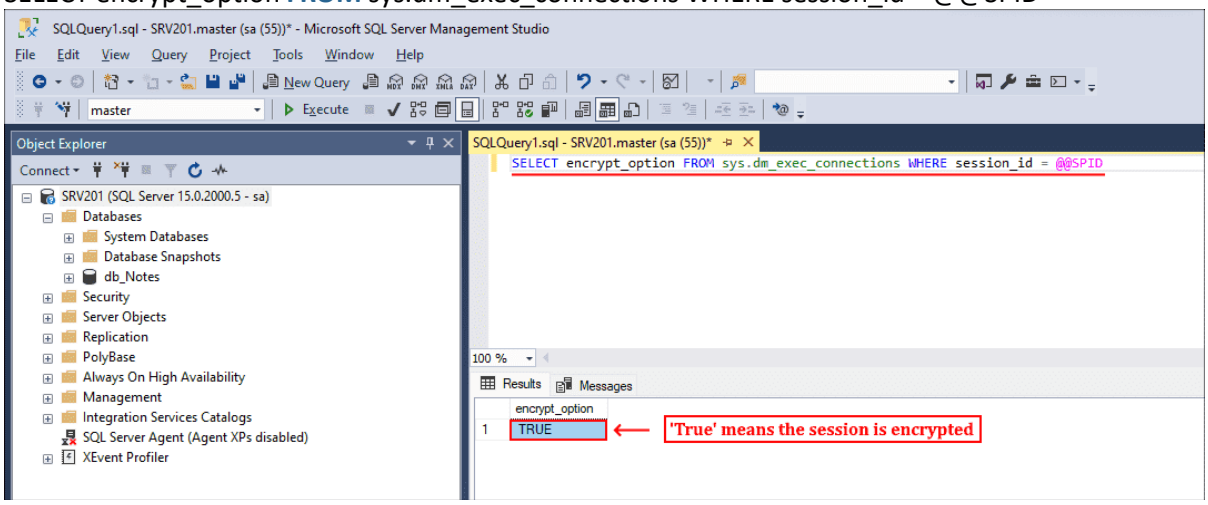
Viewing the subjects alternative name field of the certificate

Verify that TLS encryption is working ^

Now, you might be wondering, how would I know if communication between my SQL Server and client application (e.g., SQL Server Management Studio) is really encrypted?

To determine this, you can execute the following query in SSMS:

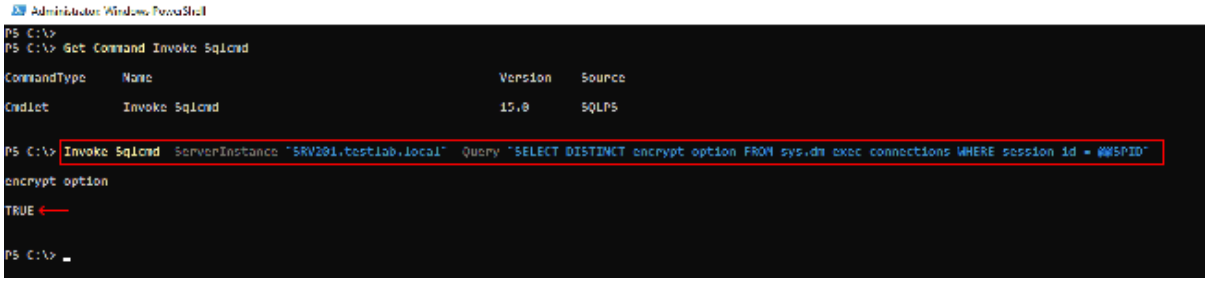
```
SELECT encrypt_option FROM sys.dm_exec_connections WHERE session_id = @@SPID
```



Check connection encryption using SQL Server Management Studio

It returns a Boolean value (either *True* or *False*), where *True* indicates that the connection is encrypted. If you have the SQL Server PowerShell module installed, you could also use the *Invoke-Sqlcmd* command, as shown below:

```
Invoke-Sqlcmd -ServerInstance "SRV201.testlab.local" -Query "SELECT DISTINCT encrypt_option FROM sys.dm_exec_connections WHERE session_id = @@SPID"
```



Check connection encryption using the Invoke Sqlcmd command

Alternatively, you could also use a packet sniffing tool like Wireshark to verify that the connection is encrypted.

I hope you have successfully enabled TLS encryption on your SQL Server instance. Remember that communication encryption is absolutely essential if your applications connect through untrusted networks (such as the Internet) to a SQL Server instance.