21

Configuring Transport Layer Security Encryption

Use Transport Layer Security (TLS), a secure and standard protocol, to encrypt your database client and server connections.

- Transport Layer Security (TLS) and the Oracle Database
  TLS secures connections between the Oracle Database client and server. The database server can also connect to other databases and other services using TLS version 1.3 (the default) or 1.2. This chapter will primarily focus on configuring TLS between the Oracle Database client and server.
- Configuring TLS for the Oracle Database and Client
  This topic describes the three most common TLS configurations. More advanced and optional configurations are described later in this chapter.
- Advanced and Optional Configurations
  Oracle Database 23ai ensures that the default Transport Layer Security configuration is secure and versatile. However, Oracle provides parameters to customize and control this configuration.
- TLS and Other Oracle Products
  Transport Layer Security (TLS) can be configured when using other Oracle Database products.
- Troubleshooting the Transport Layer Security Configuration
  Common errors may occur while you use the Oracle Database Transport Layer Security.
- Migrating to and Configuring Transport Layer Security Version 1.3
  Version 1.3 of Transport Layer Security (TLS) provides stronger security and faster TLS handshakes, when compared to previous versions of TLS.

21.1 Transport Layer Security (TLS) and the Oracle Database

TLS secures connections between the Oracle Database client and server. The database server can also connect to other databases and other services using TLS version 1.3 (the default) or 1.2. This chapter will primarily focus on configuring TLS between the Oracle Database client and server.

The database client and server can be configured to use TLS depending on your requirements. There are several options to consider which are mentioned below. The primary use cases are discussed in the following topic. Advanced considerations are discussed in Advanced and Optional Configurations.

Configuring a client-server TLS connection requires the database server to have a wallet. The server wallet includes the private key, the signed user certificate, the root of trust certificate and any intermediate certificates for the database server user certificate.

The TLS wallet on the database server must be stored under the `WALLET_ROOT` location. (The parameter `WALLET_LOCATION` is deprecated for use with Oracle Database 23ai for the Oracle Database server. It is not deprecated for use with the Oracle Database client or listener.) Create a directory for TLS under `WALLET_ROOT`, so it looks like `WALLET_ROOT/<PDB GUID>/`

`tls`. Each container (including CDB root) will have its own TLS wallet, there's no configuration to have a single wallet work for more than one or all containers when using `WALLET_ROOT`.

When configuring TLS between the database client and server there are several options to consider:

- Self-signed Certificate vs Public Certificate Authority (CA) Signed Certificate
  Determine whether a self-signed certificate or a public certificate authority signed certificate is appropriate for your database configuration.
- One-way TLS vs Mutual TLS
  Determine if one-way TLS or Mutual TLS (mTLS) is appropriate for your database configuration.
- TLS With or Without a Client Wallet
  Determine if using a client wallet is appropriate for your database configuration.
- Certificate DN Matching
  Determine if certificate DN matching is appropriate for your database configuration.

21.1.1 Self-signed Certificate vs Public Certificate Authority (CA) Signed Certificate

Determine whether a self-signed certificate or a public certificate authority signed certificate is appropriate for your database configuration.

Self-signed certificate: Having a self-signed root certificate is a common practice for internally facing IT resources since you can create these yourself and it's free. The resource (in our case, the database server) will use a self-signed root certificate to sign its own database server certificate. The server certificate and self-signed root certificate are stored in the database server wallet. For the database client to be able to trust the database server certificate, a copy of the self-signed root certificate will also be needed by the client. This self-signed root certificate can be stored in a client-side wallet or installed in the client system default certificate store. The system certificate store locations for all OS are mentioned in Oracle Wallet Search Order.

Before the session is established, the database client will check if the server certificate has been signed by the same root certificate installed on the client. Storing root trust certificate in the client system default certificate store is helpful since it can also be used by other applications and browsers in the client machine. If your company uses self-signed certificates, the root trust certificate may already be installed in all the client default trust stores.

Public certificate authority (CA): A CA-signed certificate is signed by a third-party, publicly trusted certificate authority (CA). Some examples of public certificate authorities are Symantec, DigiCert, Thawte, GeoTrust, GlobalSign, GoDaddy, and Entrust. These entities are responsible for validating the person or organization that requests each certificate.

Using a public root of trust certificate authority has some advantages in that the root trust certificate is most likely already stored in the client system default certificate store. There is no extra step for the client to store the root trust certificate if it is from a public certificate authority. The disadvantage is that this normally has a payment to a third party certificate authority.

21.1.2 One-way TLS vs Mutual TLS

Determine if one-way TLS or Mutual TLS (mTLS) is appropriate for your database configuration.

One-way TLS: One-way TLS is a server-verified encrypted channel using the TLS protocol. In the standard TLS session, only the server provides a certificate to the client to authenticate itself. The client doesn't need to have a separate client certificate to authenticate itself to the server (similar to how HTTPS sessions are established). While the database server requires a wallet to store the server user certificate and private key, the database client only needs to access the trusted CA root certificate to validate the server user certificate is signed by a trusted CA root certificate. Depending on the OS platform and the database client, the trusted CA root certificate could be in the local default certificate system store or in a client wallet. One-way TLS is the most common TLS configuration and detailed configuration steps can be found in Configuring TLS Using a Public Certificate Authority Root of Trust for the Database Server Certificate.

Two-way TLS (also called Mutual TLS, mTLS): In mTLS, both the client and server present their user certificates to each other. In most cases, the same CA root certificate will have signed both of these certificates so the same root CA certificate can be used with the database server and client to authenticate the other certificate. mTLS when used in this manner is used to encrypt the link between the database and the client, and also validate both the database and the client's certificate. Database user authentication is done separately, for example, using a database username and password to authenticate the user in addition to establishing the mTLS encrypted link. A principal (human or application) can also use the client side user certificate as it's authentication mechanism. This is called PKI certificate authentication and is covered in Configuring PKI Certificate Authentication. In this case, the user certificate does double duty - establish the mTLS connection and PKI certificate authentication to the database. For detailed configuration steps for mTLS see Mutual Transport Layer Security (mTLS).

21.1.3 TLS With or Without a Client Wallet

Determine if using a client wallet is appropriate for your database configuration.

Client with a wallet: When using mTLS, a client certificate is required. The client certificate must be stored in the client wallet or Microsot Certificate Store (MCS) in Windows. The wallet must also store the trusted CA root certificate along with the required intermediate certificates.

Client without a wallet: Clients can be configured without a wallet when using TLS under these conditions:

1. One-way TLS is being configured where the client does not have its own certificate.
2. The root certificate that signed the database server certificate is stored in the system default certificate store. If the server certificate is signed by a public certificate authority, the root certificate will most likely already be there. If a self-signed certificate was used to sign the server certificate, this self-signed certificate would need to be installed in the system default certificate store to avoid using a client wallet.
3. This is only applicable to Linux and Windows clients. This works natively with Windows MCS and the native Linux keystore. On non-Windows and non-Linux OS clients, the OCI-C client will look for a PEM file stored in several locations described in Oracle Wallet Search Order.

21.1.4 Certificate DN Matching

Determine if certificate DN matching is appropriate for your database configuration.

> **Tip:**
>
> Oracle recommends using this option when configuring a TLS session.

The DN certificate match parameters are only used by the database client. When DN certificate match is enabled, the client checks information on the server certificate (common name (CN), distinguished name (DN), subject alternate names (SAN)) and compares it with the information in the connect string or sqlnet.ora. If there's a match, it means that the database server is the expected server that the client wanted to connect with. If there's no match, the client rejects the connection attempt since the server is not the intended server. Configuring TLS without checking for a partial or full DN match checks that the server certificate has not expired and has been signed by a known certificate authority. DN certificate match takes it one step further and makes sure the client is talking to the expected server. There are 2 sub-options for DN certificate match: Partial DN match and Full DN match.

- **Partial DN match:** In `SQLNET.ora` or in the connect string, specify `SSL_SERVER_DN_MATCH=YES`. Partial DN match will check the `HOST` parameter in the connect string to see if there's a match with the CN, DN, or SAN names. There has to be a match for the connection to be successful.
- **Full DN match:** In addition to setting `SSL_SERVER_DN_MATCH=YES`, you must also set `SSL_SERVER_CERT_DN=<certificate DN>` to force a full DN match. This allows you to continue to use DN certificate match when your `HOST` value needs to be an IP address or something other than the names available in the certificate.

21.2 Configuring TLS for the Oracle Database and Client

This topic describes the three most common TLS configurations. More advanced and optional configurations are described later in this chapter.

- About Configuring TLS for the Oracle Database
  The three most common TLS configurations are described in detail in this topic.
- Configuring TLS Using a Public Certificate Authority Root of Trust for the Database Server Certificate
  Before you can configure TLS without using client wallets, you must first create the server wallet and ensure that the database and listener are properly configured.
- Configuring TLS with a Self-Signed Root Certificate
  Using a self-signed root certificate is very similar to the above use case, except you must create a root wallet and sign the database certificate with the self-signed root certificate.
- Configuring TLS Connection With a Client Wallet
  A client wallet is sometimes required when configuring TLS with a public or self-signed CA trust certificate.
- Enabling Distinguished Name (DN) Matching
  DN matching allows a connection to the Oracle Database server when the server certificate name or DN matches what the client expects.

21.2.1 About Configuring TLS for the Oracle Database

The three most common TLS configurations are described in detail in this topic.

The first decision is to use a self-signed certificate root of trust or a public CA root of trust. Once you make that decision, Oracle recommends using TLS without a wallet if your

environment supports this and is allowed by your security policies. This greatly simplifies managing database clients. Start your configurations with the minimum set of mandatory parameters. And then once you are successful, add the recommended parameters and any optional parameters one by one.

The following parameters are used in the following configurations in this topic.

Table 21-1 Mandatory and Recommended parameters to configure one-way TLS

Parameter	Description	Server (Defined in `sqlnet.ora`)	Listener (Defined in `listener.ora`)	Static Client (Defined in `sqlnet.ora`)	Dynamic Client (Defined in the connect string)
`WALLET_ROOT`	Database server system parameter (replaces `WALLET_LOCATION`)	Required	No	No	No
`WALLET_LOCATION`	Specifies wallet location if required	No	Required	Optional	Optional
`Protocol=tcps`	Enables TLS connection	No	Required	No	Required
`SSL_CLIENT_AUTHENTICATION`	Disable to allow 1-way TLS	Required	Required	Optional	Optional
`SSL_SERVER_DN_MATCH`	Enables partial or full DN matching	No	No	Recommended	Recommended
`SSL_SERVER_CERT_DN`	Use if full DN matching is required	No	No	No	Optional

`WALLET_ROOT` and `WALLET_LOCATION` Parameters

For Oracle Database server, Oracle recommends that you use the `WALLET_ROOT` system parameter instead of using `WALLET_LOCATION`.

The parameter `WALLET_LOCATION` is deprecated for use with Oracle Database 23ai for the Oracle Database server. It is not deprecated for use with the Oracle Database client or listener. The TLS wallet location for a PDB is `WALLET_ROOT/<PDB GUID>/tls`.

`WALLET_LOCATION` must be used by the listener to find its wallet. Oracle recommends using the same wallet for the listener and the server for DN matching. DN matching is used by the client to verify that it is connecting to the expected server, and the client checks both the listener and the server certificates.

`Protocol` Parameter

The `Protocol` must be set to `tcps` with the client and listener. The listener sets this as part of the service connect string. The client sets this in the connect string.

`SSL_CLIENT_AUTHENTICATON` Parameter

`SSL_CLIENT_AUTHENTICATON` must be set to `FALSE` for the database server and the listener to allow TLS traffic (vs mTLS) to connect to the listener and the server. This is optional for the client and depends if the client already has a wallet with a client-side user certificate that is used for other connections.

DN Matching

Oracle recommends using DN matching. However, add these settings once you have successfully confirmed a TLS connection.

The most common TLS configurations for the Oracle Database are:

- Configuring TLS Using a Public Certificate Authority Root of Trust for the Database Server Certificate
- Configuring TLS with a Self-Signed Root Certificate
- Configuring TLS Connection With a Client Wallet

21.2.2 Configuring TLS Using a Public Certificate Authority Root of Trust for the Database Server Certificate

Before you can configure TLS without using client wallets, you must first create the server wallet and ensure that the database and listener are properly configured.

Create the Server and Listener Wallet

To get a certificate signed by a publicly signed certificate authority, you must create the database server and listener wallet and export a certificate signing request (CSR).

1. Login to the host where the database is installed.
2. Create the wallet.

   ```
   orapki wallet create -wallet <wallet location> -pwd <wallet password> -
   auto_login
   ```

3. Add the trusted root certificate to the wallet (get this from your certificate administrator).

   ```
   orapki wallet add -wallet <wallet location> -trusted_cert -cert <trusted
   root certificate location>/rootCA.crt -pwd <wallet password>
   ```

4. Create a private key and certificate request in the wallet.

   ```
   orapki wallet add -wallet <wallet location> -keysize 2048 -dn
   <certificate_dn> -pwd <wallet password>
   ```

5. Export the certificate request to get it signed.

   ```
   orapki wallet export -wallet <wallet location> -dn <certificate_dn> -
   request <certificate signing request location>/<file_name>.csr -pwd
   <wallet password>
   ```

6. Display the contents of the wallet.

   ```
   orapki wallet display -wallet <wallet_location>
   ```

   There will be an entry under Requested Certificates.

7. View the contents of the CSR (certificate signing request) file.

   ```
   cat <certificate_signing_request_location>/<file_name>.csr
   ```

8. Send the CSR file to your certificate administrator to have it signed by the root certificate authority (CA) or an intermediate CA.
9. Import the signed database server user certificate into the database wallet.

   ```
   orapki wallet add -wallet <wallet location> -user_cert -cert <signed
   certificate location>/<file_name_signed>.crt -pwd <wallet password>
   ```

10. Display the contents of the wallet:

    ```
    orapki wallet display -wallet <wallet location>
    ```

11. Ensure that the database server user certificate is now displayed under `User Certificates`.

    The wallet you will use for the database server and listener is now ready to be deployed for use.

Set `WALLET_ROOT` and deploy the database server wallet

1. Check to see if `WALLET_ROOT` already exists. Login as a user with privileges to check system parameters and run:

   ```
   SHOW PARAMETER WALLET_ROOT
   ```

   If `WALLET_ROOT` is not already setup, run the next command to create `WALLET_ROOT`.
2. Create `WALLET_ROOT`, a system parameter. Run the following SQL command:

   ```
   alter system set wallet_root = '<wallet_root_directory>' scope=spfile;
   ```

3. Reboot the database.
4. Show the modified `wallet_root` parameter. Run the following SQL command:

   ```
   show parameter wallet_root;
   ```

5. If the TLS directory does not yet exist under `WALLET_ROOT`, create a directory for TLS under your `WALLET_ROOT` PDB directory in the operating system.

   ```
   mkdir -p -v <wallet_root_directory>/<PDB GUID>/tls
   ```

   You can find the PDB GUID for your PDB by running the following SQL command:

   ```
   select guid from v\$containers;
   ```

6. Change ownership of the directory.

   ```
   sudo chown oracle:oinstall -R -v <wallet_root_directory>/<PDB GUID>/tls
   ```

7. Copy the database server ewallet.p12 and the cwallet.sso files to this new tls directory.

Perform this command from the same directory where the wallets were created:

```
cp ./ewallet.p12 ./cwallet.sso <wallet_root_directory>/<PDB GUID>/tls
```

Database server configuration for TLS

1. Log in to the server where the Oracle database resides.
2. Check that `SSL_CLIENT_AUTHENTICATION` in the `sqlnet.ora` file is set to `FALSE` as this enables one-way TLS:

   By default, the `sqlnet.ora` file is located in the `$ORACLE_HOME/network/admin` directory or in the location set by the `TNS_ADMIN` environment variable.
   When using read-only Oracle home, the default location for `sqlnet.ora` is `$ORACLE_HOME/network/admin`.

   ```
   SSL_CLIENT_AUTHENTICATION=FALSE
   ```

   You may set this to `OPTIONAL` instead which enables both TLS and mTLS and is dependent on whether the client sends the client user certificate.

Listener configuration for TLS

1. Check the `PROTOCOL` parameter in the `listener.ora` file to ensure TLS is specified.

   By default, `listener.ora` is located in the `$ORACLE_HOME/network/admin` directory.

   The parameter `PROTOCOL=tcps` tells the listener to only use TLS (or mTLS) for database connections.

   For example:

   ```
   LISTENER = (ADDRESS=(PROTOCOL=tcps)(HOST=<host_name>)(PORT=1522))
   ```

2. Ensure that the listener wallet exists in the location of the `WALLET_LOCATION` parameter in the `listener.ora` file. Use the same wallet as you did for the database server.

   ```
   WALLET_LOCATION=
       (SOURCE=
           (METHOD=file)
           (METHOD_DATA=
               (DIRECTORY=$WALLET_DIR/<pdb guid>/tls)))
   ```

   If the listener is on the same server as the database server and the server TLS wallet is in the default location, set the listener `WALLET_LOCATION` to the same location. Alternatively, the server wallet can be copied to a different location for the listener.

   If you set the `SSL_SERVER_DN_MATCH` parameter to `TRUE` for DN matching (partial or full DN match), then the hostname or DN check will happen against both the listener certificate and the server certificate. They don't have to be the same certificate, but matching will be done with both certificates.

3. Ensure the `SSL_CLIENT_AUTHENTICATION` parameter is set to `FALSE` in `listener.ora` file to disable mutual TLS.

   ```
   SSL_CLIENT_AUTHENTICATION=FALSE
   ```

> **Note:**
>
> If the listener supports multiple databases, some with one-way TLS and some with mTLS, then set `SSL_CLIENT_AUTHENTICATION=OPTIONAL`.

Client Configuration for TLS

Configure Client Connect String for TLS

Add the parameter `protocol=tcps` in the connect string to enforce TLS from the client. The connection will use TLS from the client to the listener.

> **Note:**
>
> This parameter is not available in `sqlnet.ora`.

```
(description=
    (address=
        (protocol=tcps)
        (port=1522)
        (host=example.com))
    (connect_data=
        (service_name=dbservicename.example.com)))
```

(Optional) Set `SSL_CLIENT_AUTHENTICATON` for the Client

- If you have a client-side user certificate, but don't want to use it for mTLS, then you must complete this step.
- If you don't have a client-side user certificate, you can skip this step as the client will go ahead and make a one-way TLS connection regardless of this parameter setting.

1. Log in to the client for the Oracle database.
2. Set `SSL_CLIENT_AUTHENTICATION` in the `sqlnet.ora` file to `FALSE`.

   ```
   SSL_CLIENT_AUTHENTICATION=FALSE
   ```

   Setting this parameter in `sqlnet.ora` to `FALSE`, will block sending a client side user certificate for all the connections. You can override this for a particular connection by setting `SSL_CLIENT_AUTHENTICATION=TRUE` in the connection string in `tnsnames.ora` so that connection will use the client-side user certificate.

   The connection string parameter will take precedence over the `sqlnet.ora` parameter setting. This setting is optional and only required if you have a client-side user certificate and you don't want to use it for an mTLS connection.
3. In order to preserve existing mTLS connections that use the client-side wallet and user certificate and also to establish one-way TLS connection without using the user certificate, set `SSL_CLIENT_AUTHENTICATION=TRUE`, which is the default setting, in `sqlnet.ora`. Then for every connection that you want to use without a client-side user wallet, add `SSL_CLIENT_AUTHENTICATION=FALSE` in the connect string.

Connect to the database

Connect to the database using the connection name with the tcps protocol.

```
sqlplus <user_name>@<PDB_name>
```

Verify the connection

1. Run the following command:

   ```
   select sys_context ('userenv','NETWORK_PROTOCOL') from dual;
   ```

   This will show 'tcps' if TLS is enabled and 'tcp' if TLS is not enabled.
2. Run the following command:

   ```
   select sys_context ('userenv','TLS_VERSION') from dual;
   ```

   This will show the TLS protocol for the connection ending at the database server.
3. Run the following command:

   ```
   select sys_context ('userenv','TLS_CIPHERSUITE') from dual;
   ```

   This will show the TLS ciphersuite for the connection ending at the database server.

21.2.3 Configuring TLS with a Self-Signed Root Certificate

Using a self-signed root certificate is very similar to the above use case, except you must create a root wallet and sign the database certificate with the self-signed root certificate.

Create the Root Wallet

1. Create the root wallet:

   ```
   orapki wallet create -wallet <root wallet directory> -pwd <root wallet
   password> -auto_login
   ```
2. View the contents of the wallet, it should be empty:

   ```
   orapki wallet display -wallet <root wallet directory>
   ```
3. Create the self-signed certificate for the root CA wallet:

   ```
   orapki wallet add -wallet <root wallet directory> -dn <certificate_DN> -
   keysize 2048 -sign_alg sha256 -self_signed -validity 365 -pwd <root wallet
   password>
   ```
4. The directory should now have cwallet.sso and ewallet.p12 files:

   ```
   ls -l <root wallet directory>
   ```

5. View the contents of the wallet, it should have a user and a trusted certificate:

```
orapki wallet display -wallet <root wallet directory>
```

6. Export the root CA trusted certificate for use in creating the DB wallet:

```
orapki wallet export -wallet <root wallet directory> -dn <certificate_DN> -
cert <root wallet directory>/rootCA.crt -pwd <root wallet password>
```

7. View the contents of the rootCA.crt file:

```
cat <root wallet directory>/rootCA.crt
```

Create the Server and Listener Wallet

To get a certificate signed by the self-signed root certificate, follow the same steps as in the prior use case, where you create the wallets and export a certificate signing request (CSR).

1. Login to the host where the database is installed.
2. Create the wallet.

```
orapki wallet create -wallet <wallet location> -pwd <wallet password> -
auto_login
```

3. Add the trusted root certificate to the wallet (get this from your certificate administrator).

```
orapki wallet add -wallet <wallet location> -trusted_cert -cert <trusted
root certificate location>/rootCA.crt -pwd <wallet password>
```

4. Create a private key and certificate request in the wallet.

```
orapki wallet add -wallet <wallet location> -keysize 2048 -dn
<certificate_dn> -pwd <wallet password>
```

5. Export the certificate request to get it signed.

```
orapki wallet export -wallet <wallet location> -dn <certificate_dn> -
request <certificate signing request location>/<file_name>.csr -pwd
<wallet password>
```

6. Display the contents of the wallet.

```
orapki wallet display -wallet <wallet_location>
```

There will be an entry under Requested Certificates.

7. View the contents of the CSR (certificate signing request) file.

```
cat <certificate_signing_request_location>/<file_name>.csr
```

Sign the database server certificate signing request (CSR) file

1. Sign the CSR using the self-signed root wallet:

   ```
   orapki cert create -wallet <root wallet directory> -request <CSR
   directory>/example.csr -cert <wallet location>/example-signed.crt -
   validity 365 -sign_alg sha256 -pwd <root wallet password>
   ```

2. View the signed server user certificate:

   ```
   cat <wallet location>/example-signed.crt
   ```

3. Import the signed database server user certificate into the database wallet.

   ```
   orapki wallet add -wallet <wallet location> -user_cert -cert <signed
   certificate location>/<file_name_signed>.crt -pwd <wallet password>
   ```

4. Display the contents of the wallet:

   ```
   orapki wallet display -wallet <wallet location>
   ```

5. Ensure that the database server user certificate is now displayed under `User Certificates`.

   The wallet you will use for the database server and listener is now ready to be deployed for use.

Set `WALLET_ROOT` and deploy the database server wallet

1. Check to see if `WALLET_ROOT` already exists. Login as a user with privileges to check system parameters and run:

   ```
   SHOW PARAMETER WALLET_ROOT
   ```

   If `WALLET_ROOT` is not already setup, run the next command to create `WALLET_ROOT`.

2. Create `WALLET_ROOT`, a system parameter. Run the following SQL command:

   ```
   alter system set wallet_root = '<wallet_root_directory>' scope=spfile;
   ```

3. Reboot the database.
4. Show the modified `wallet_root` parameter. Run the following SQL command:

   ```
   show parameter wallet_root;
   ```

5. If the TLS directory does not yet exist under `WALLET_ROOT`, create a directory for TLS under your `WALLET_ROOT` PDB directory in the operating system.

   ```
   mkdir -p -v <wallet_root_directory>/<PDB GUID>/tls
   ```

You can find the PDB GUID for your PDB by running the following SQL command:

```
select guid from v\$containers;
```

6. Change ownership of the directory.

```
sudo chown oracle:oinstall -R -v <wallet_root_directory>/<PDB GUID>/tls
```

7. Copy the database server ewallet.p12 and the cwallet.sso files to this new tls directory.

   Perform this command from the same directory where the wallets were created:

```
cp ./ewallet.p12 ./cwallet.sso <wallet_root_directory>/<PDB GUID>/tls
```

Database server configuration for TLS

1. Log in to the server where the Oracle database resides.
2. Check that `SSL_CLIENT_AUTHENTICATION` in the `sqlnet.ora` file is set to `FALSE` as this enables one-way TLS:

   By default, the `sqlnet.ora` file is located in the `$ORACLE_HOME/network/admin` directory or in the location set by the `TNS_ADMIN` environment variable.
   When using read-only Oracle home, the default location for `sqlnet.ora` is `$ORACLE_HOME/network/admin`.

```
SSL_CLIENT_AUTHENTICATION=FALSE
```

   You may set this to `OPTIONAL` instead which enables both TLS and mTLS and is dependent on whether the client sends the client user certificate.

Listener configuration for TLS

1. Check the `PROTOCOL` parameter in the `listener.ora` file to ensure TLS is specified.

   By default, `listener.ora` is located in the `$ORACLE_HOME/network/admin` directory.

   The parameter `PROTOCOL=tcps` tells the listener to only use TLS (or mTLS) for database connections.

   For example:

```
LISTENER = (ADDRESS=(PROTOCOL=tcps)(HOST=<host_name>)(PORT=1522))
```

2. Ensure that the listener wallet exists in the location of the `WALLET_LOCATION` parameter in the `listener.ora` file. Use the same wallet as you did for the database server.

```
WALLET_LOCATION=
    (SOURCE=
        (METHOD=file)
        (METHOD_DATA=
            (DIRECTORY=$WALLET_DIR/<pdb guid>/tls)))
```

   If the listener is on the same server as the database server and the server TLS wallet is in the default location, set the listener `WALLET_LOCATION` to the same location. Alternatively, the server wallet can be copied to a different location for the listener.

If you set the `SSL_SERVER_DN_MATCH` parameter to `TRUE` for DN matching (partial or full DN match), then the hostname or DN check will happen against both the listener certificate and the server certificate. They don't have to be the same certificate, but matching will be done with both certificates.

3. Ensure the `SSL_CLIENT_AUTHENTICATION` parameter is set to `FALSE` in `listener.ora` file to disable mutual TLS.

```
SSL_CLIENT_AUTHENTICATION=FALSE
```

> **Note:**
>
> If the listener supports multiple databases, some with one-way TLS and some with mTLS, then set `SSL_CLIENT_AUTHENTICATION=OPTIONAL`.

Client Configuration for TLS

Add the self-signed trusted root certificate to the client system default keystore

On the database client operating systems, you need to add the self-signed trusted root certificate to the client system's default keystore. If your company is using a corporate self-signed trusted root certificate, this may already be done for you.
The Oracle Database thick clients (OCI-C) work natively with the Windows and Linux system default stores. On other operating systems, the Oracle Database client will search the directory locations listed below for a PEM file. If your PEM file for your OS is in a different location, you can either copy the PEM file to one of the searched locations or create a symlink to a searched location. Follow the directions for your OS to add the new trust certificate to your system certificate store (PEM file). We include the directions for doing that for Microsoft Windows and RHEL/Oracle Linux.

1. Export the root CA trusted certificate from the root wallet.

```
orapki wallet export -wallet <root wallet location> -dn <certificate_DN> -
cert <root wallet location>/rootCA.crt -pwd <root wallet password>
```

2. Append the exported database trust certificate to the system's default certificate store.
   - For Windows, use the Microsoft Management Console (mmc) to import the trusted root certificate to the Microsoft Certificate Store (MCS)
   - For RHEL/Oracle Linux, the default system store is at `/etc/pki/tls/cert.pem`. To import the new root certificate to this PEM file, do the following:
     a. Add your new certificate to: `/etc/pki/ca-trust/source/anchors/`
     b. Run the following:

        ```
        sudo update-ca-trust extract
        ```

     c. Delete the standalone root certificate:

        ```
        rm -v <root wallet location>/rootCA.crt
        ```

   - For the remaining Linux operating systems, the PEM file can be found at:
     – RHEL/Oracle Linux: `/etc/pki/tls/cert.pem`

  - Debian/Ubuntu/Gentoo: `/etc/ssl/certs/ca-certificates.crt`
  - Fedora/RHEL: `/etc/pki/tls/certs/ca-bundle.crt`
  - OpenSUSE: `/etc/ssl/ca-bundle.pem`
  - OpenELEC: `/etc/pki/tls/cacert.pem`
  - CentOS/RHEL7: `/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem`
  - Alpine Linux: `/etc/ssl/cert.pem`

  Follow your OS instructions for adding a new certificate to your existing PEM file.

- For non-Linux and non-Windows systems, if the PEM file is not in one of the locations listed above for Linux systems, then you must either copy the PEM file to one of these default Linux locations or create a symlink from the PEM file to one of these locations. The file must be a PEM file.

> **Note:**
>
> You cannot change the default location of the certificate store.

Configure Client Connect String for TLS

Add the parameter `protocol=tcps` in the connect string to enforce TLS from the client. The connection will use TLS from the client to the listener.

> **Note:**
>
> This parameter is not available in `sqlnet.ora`.

```
(description=
    (address=
        (protocol=tcps)
        (port=1522)
        (host=example.com))
    (connect_data=
        (service_name=dbservicename.example.com)))
```

(Optional) Set `SSL_CLIENT_AUTHENTICATON` for the Client

- If you have a client-side user certificate, but don't want to use it for mTLS, then you must complete this step.
- If you don't have a client-side user certificate, you can skip this step as the client will go ahead and make a one-way TLS connection regardless of this parameter setting.

1. Log in to the client for the Oracle database.
2. Set `SSL_CLIENT_AUTHENTICATION` in the `sqlnet.ora` file to `FALSE`.

   ```
   SSL_CLIENT_AUTHENTICATION=FALSE
   ```

Setting this parameter in `sqlnet.ora` to `FALSE`, will block sending a client side user certificate for all the connections. You can override this for a particular connection by setting `SSL_CLIENT_AUTHENTICATION=TRUE` in the connection string in `tnsnames.ora` so that connection will use the client-side user certificate.

The connection string parameter will take precedence over the `sqlnet.ora` parameter setting. This setting is optional and only required if you have a client-side user certificate and you don't want to use it for an mTLS connection.

3. In order to preserve existing mTLS connections that use the client-side wallet and user certificate and also to establish one-way TLS connection without using the user certificate, set `SSL_CLIENT_AUTHENTICATION=TRUE`, which is the default setting, in `sqlnet.ora`. Then for every connection that you want to use without a client-side user wallet, add `SSL_CLIENT_AUTHENTICATION=FALSE` in the connect string.

Connect to the database

Connect to the database using the connection name with the tcps protocol.

```
sqlplus <user_name>@<PDB_name>
```

Verify the connection

1. Run the following command:

   ```
   select sys_context ('userenv','NETWORK_PROTOCOL') from dual;
   ```

   This will show 'tcps' if TLS is enabled and 'tcp' if TLS is not enabled.

2. Run the following command:

   ```
   select sys_context ('userenv','TLS_VERSION') from dual;
   ```

   This will show the TLS protocol for the connection ending at the database server.

3. Run the following command:

   ```
   select sys_context ('userenv','TLS_CIPHERSUITE') from dual;
   ```

   This will show the TLS ciphersuite for the connection ending at the database server.

21.2.4 Configuring TLS Connection With a Client Wallet

A client wallet is sometimes required when configuring TLS with a public or self-signed CA trust certificate.

A client wallet for a TLS connection includes the trust certificate for the certificate authority that signed the database server certificate. Only the root of trust certificate is required. Intermediate certificates are not required.

Using a client wallet is required if you cannot use the system's default certificate store.

1. Create the client wallet.

   ```
   orapki wallet create -wallet <wallet_location> -pwd <wallet_password> -
   auto_login
   ```

2. Get the CA trusted certificate. This may already be available in a file or you may need to export it from the root certificate wallet or a database server wallet.

   ```
   orapki wallet export -wallet <wallet_location> -dn <certificate_dn> -cert
   <certificate_filename>
   ```

   For more information see orapki Utility Commands Summary.

3. Add the CA trusted certificate into the client wallet.

   ```
   orapki wallet add -wallet <wallet_location> -trusted_cert -cert
   <certificate_filename>
   ```

4. Move or copy the client wallet to the desired location.
5. Update `sqlnet.ora` to add `WALLET_LOCATION` for the client wallet.

   This will be used by all client connections unless this is overridden by the connect string parameter `WALLET_LOCATION`. When `WALLET_LOCATION` is not set in `sqlnet.ora` or the connect string, then the client will check the system's default certificate store.

   ```
   WALLET_LOCATION=
       (SOURCE=
           (METHOD=file)
           (METHOD_DATA=
               (DIRECTORY=/etc/oracle/wallets/databases)))
   ```

   See WALLET_LOCATION in the *Oracle Database Net Services Reference* guide for more information.

(Optional) Set `SSL_CLIENT_AUTHENTICATON` for the Client

- If you have a client-side user certificate, but don't want to use it for mTLS, then you must complete this step.
- If you don't have a client-side user certificate, you can skip this step as the client will go ahead and make a one-way TLS connection regardless of this parameter setting.

1. Log in to the client for the Oracle database.
2. Set `SSL_CLIENT_AUTHENTICATION` in the `sqlnet.ora` file to `FALSE`.

   ```
   SSL_CLIENT_AUTHENTICATION=FALSE
   ```

   Setting this parameter in `sqlnet.ora` to `FALSE`, will block sending a client side user certificate for all the connections. You can override this for a particular connection by setting `SSL_CLIENT_AUTHENTICATION=TRUE` in the connection string in `tnsnames.ora` so that connection will use the client-side user certificate.

   The connection string parameter will take precedence over the `sqlnet.ora` parameter setting. This setting is optional and only required if you have a client-side user certificate and you don't want to use it for an mTLS connection.

3. In order to preserve existing mTLS connections that use the client-side wallet and user certificate and also to establish one-way TLS connection without using the user certificate, set `SSL_CLIENT_AUTHENTICATION=TRUE`, which is the default setting, in `sqlnet.ora`. Then for every connection that you want to use without a client-side user wallet, add `SSL_CLIENT_AUTHENTICATION=FALSE` in the connect string.

Connect to the database

Connect to the database using the connection name with the tcps protocol.

```
sqlplus <user_name>@<PDB_name>
```

Verify the connection

1. Run the following command:

   ```
   select sys_context ('userenv','NETWORK_PROTOCOL') from dual;
   ```

   This will show 'tcps' if TLS is enabled and 'tcp' if TLS is not enabled.
2. Run the following command:

   ```
   select sys_context ('userenv','TLS_VERSION') from dual;
   ```

   This will show the TLS protocol for the connection ending at the database server.
3. Run the following command:

   ```
   select sys_context ('userenv','TLS_CIPHERSUITE') from dual;
   ```

   This will show the TLS ciphersuite for the connection ending at the database server.

21.2.5 Enabling Distinguished Name (DN) Matching

DN matching allows a connection to the Oracle Database server when the server certificate name or DN matches what the client expects.

> **Tip:**
>
> Oracle strongly recommends using either partial or full DN matching so the client connects to the correct host.

When DN matching is enabled, the listener certificate and the database server certificate will both be checked against the certificate expected by the client. Without using DN matching, any server certificate signed by the same or valid public CA will be accepted by the client to establish the TLS session.

It is recommended to first successfully configure TLS in a test environment prior to setting up DN matching. See Configuring TLS Using a Public Certificate Authority Root of Trust for the Database Server Certificate.

To enable DN Matching:

1. Set the `SSL_SERVER_DN_MATCH` parameter to `TRUE` in the `sqlnet.ora` file:

   ```
   SSL_SERVER_DN_MATCH = TRUE
   ```

   The `sqlnet.ora` file will look similar to:

   ```
   SSL_CLIENT_AUTHENTICATION = FALSE
   WALLET_LOCATION =
       (SOURCE=
       (METHOD=File)
       (METHOD_DATA=
           (DIRECTORY=wallet_location)))
   SSL_SERVER_DN_MATCH = TRUE
   ```

   > **Note:**
   >
   > Only completing this step will result in partial DN matching. Perform step three to establish full DN matching.

   Partial DN matching will check the host parameter value in the connect string against the certificate's common name (CN). If a match isn't found, the client will then compare the host parameter value against the entries in the certificate's Subject Alternate Name (SAN) field. If there are no matches, the connection will be refused.

2. Check the host name parameter in the connect string in `tnsnames.ora` against the common name (CN) of the certificate DN string and the hostnames listed in the Subject Alternate Name (SAN) field. The connect string host name needs to match for partial DN match to succeed.
   The `tnsnames.ora` file can be located on the client or in the LDAP directory. The `tnsnames.ora` file is typically located in the setting specified by the `TNS_ADMIN` environment variable. If `TNS_ADMIN` is not set, then `tnsnames.ora` resides in the following directory locations:

   - Linux:

     ```
     $ORACLE_HOME/network/admin/
     ```

   - Windows:

     ```
     ORACLE_BASE\ORACLE_HOME\network\admin\
     ```

3. If you can't use partial DN matching, then configure full DN matching by setting the `SSL_SERVER_CERT_DN` parameter connection string in the `tnsnames.ora` file:

> **Note:**
>
> If you can't set the host value in `tnsnames.ora` or `sqlnet.ora` to the value of the certificate common name (CN) or one of the entries in the SAN field, then consider using full DN matching.
>
> Both the listener and server certificate will be checked with both partial and full DN matching. When using full DN matching, while the server and listener certificate can be different, their DN must be the same for the connection to succeed.

The `tnsnames.ora` file will look similar to:

```
finance=
(DESCRIPTION=
    (ADDRESS_LIST=
        (ADDRESS=(PROTOCOL = tcps)(HOST = finance)
        (PORT = 1575)))
    (CONNECT_DATA=
        (SERVICE_NAME= finance.us.example.com))
    (SECURITY=
        (SSL_SERVER_DN_MATCH = TRUE)
        (SSL_SERVER_CERT_DN="cn=finance,cn=OracleContext,c=us,o=example")))
```

21.3 Advanced and Optional Configurations

Oracle Database 23ai ensures that the default Transport Layer Security configuration is secure and versatile. However, Oracle provides parameters to customize and control this configuration.

> **Note:**
>
> To ensure secure configuration, Oracle recommends that only mandatory and recommended parameters are configured in your environment. When the Oracle Database client and server are configuring a connection, the most secure protocol and cipher suite that is common to both the server and client are selected. Selecting a TLS protocol or cipher suite will block clients that are unable to use that protocol or cipher suite. These configurations need to be checked during database updates and upgrades to make sure the selected values are supported after the database upgrade or update.

- Optional Parameters for Transport Layer Security
  The server-side TLS configuration is applicable to all connections serviced by the server. These are specified in the server-side configuration files sqlnet.ora for the Database server and listener.ora for the Database listener.
- Mutual Transport Layer Security (mTLS)
  In traditional Transport Layer Security (TLS), only the server authenticates to the client by presenting its certificate. With mutual Transport Layer Security (mTLS), both the server and the client present their certificates so that they are mutually authenticated.

- Oracle Wallet Location
  Certificates used for TLS are stored in the Oracle wallet. There are several default locations where the wallet can be placed. The location of the wallet can also be configured with the wallet location parameters on the client and listener. The `WALLET_ROOT` system parameter should be used for the database server wallet location.
- Enable Weak DN Matching
  The `SSL_ALLOW_WEAK_DN_MATCH` parameter control reverts the DN matching behavior to prior database versions.
- Private Key/Certificate Selection
  You can have multiple private key/certificate pairs stored in either the Oracle wallet or a system certificate store to use for certificates. This is sometimes necessary when different databases will assign different client credentials for mTLS, such as for Autonomous Database.
- Transport Layer Security Encryption Combined with Authentication Methods
  You can configure Oracle Database to use TLS concurrently with all authentication mechanisms such as database user names and passwords, RADIUS, Kerberos, PKI certificates, MS-EI, and OCI IAM.
- Specifying TLS Protocol and TLS Cipher Suites
  Oracle Database 23ai supports TLS protocol versions 1.2 and 1.3 and their associated cipher suites for Transport Layer Security (TLS).
- Certificate Validation with Certificate Revocation Lists
  Oracle provides tools that enable you to validate certificates using certificate revocation lists.

21.3.1 Optional Parameters for Transport Layer Security

The server-side TLS configuration is applicable to all connections serviced by the server. These are specified in the server-side configuration files sqlnet.ora for the Database server and listener.ora for the Database listener.

The client-side TLS configuration can be connection-specific or applied to all connections via `sqlnet.ora`. There are two ways to configure a Transport Layer Security (TLS) parameter for clients.

- **Static**: these are parameters specified in the configuration file `sqlnet.ora` and uniformly applied to all connections made by the client.
- **Dynamic**: If desired, certain TLS parameters may be specified directly in the TNS connect string to override the same or similar parameter in `sqlnet.ora`.

Table 21-2 General TLS Parameters

Parameter	Description	Server	Listener	Static Client	Dynamic Client
`HTTPS_CLIENT_AUTHENTICATION`	Specifies whether a client is authenticated using TLS for HTTPS connections	Yes	Yes	Yes	Yes
`SSL_CLIENT_AUTHENTICATION`	Specifies whether a client is authenticated using TLS or mTLS	Yes	Yes	Yes	Yes
`WALLET_LOCATION`	Specify the TLS wallet location.	Yes*	Yes	Yes	Yes

*The parameter WALLET_LOCATION is deprecated for use with Oracle Database 23ai for the Oracle Database server. It is not deprecated for use with the Oracle Database client or listener.

For Oracle Database server, Oracle recommends that you use the WALLET_ROOT system parameter instead of using WALLET_LOCATION.

Table 21-3 TLS Parameters For Selecting User Certificate

Parameter	Description	Server	Listener	Static Client	Dynamic Client
SSL_CERTIFICATE_ALIAS	Specifies the certificate based on its alias.	Yes	Yes	Yes	Yes
SSL_EXTENDED_KEY_USAGE	Specifies the certificate based on its key usage value.	Yes	Yes	Yes	Yes
SSL_CERTIFICATE_THUMBPRINT	Specifies the certificate based on its thumbprint.	Yes	Yes	Yes	Yes

> **Note:**
>
> Selecting a client-side user certificate is only applicable when working with user certificates in Windows MCS and in Oracle wallets.

Table 21-4 TLS Certificate DN Matching Parameters

Parameter	Description	Server	Listener	Static Client	Dynamic Client
SSL_ALLOW_WEAK_DN_MATCH	Allows the earlier weaker distinguished name (DN) matching behavior during server-side certificate validation	No	No	Yes	Yes
SSL_SERVER_CERT_DN	Specifies the distinguished name (DN) of the database server	No	No	No	Yes
SSL_SERVER_DN_MATCH	Enforces client-side validation of server through distinguished name (DN) matching	No	No	Yes	Yes
TLS_CERT_VALIDATION_MODE	Specifies if stricter checks as per RFC 5280 are enforced.	No	No	Yes	No

Table 21-5 TLS Protocol and Cipher Suite Selection Parameters

Parameter	Description	Server	Listener	Static Client	Dynamic Client
SSL_CIPHER_SUITES	Specifies the TLS cipher suites allowed for TLS connections	Yes	Yes	Yes	Yes
SSL_ENABLE_WEAK_CIPHERS	Enables deprecated TLS cipher suites	Yes	Yes	Yes	Yes

Table 21-5 (Cont.) TLS Protocol and Cipher Suite Selection Parameters

Parameter	Description	Server	Listener	Static Client	Dynamic Client
SSL_VERSION	Specifies the TLS protocol to use in a connection	Yes	Yes	Yes	Yes
TLS_DISABLE_VERSION	Specifies what, if any, TLS protocols are disallowed in a connection.	Yes	Yes	Yes	Yes

21.3.2 Mutual Transport Layer Security (mTLS)

In traditional Transport Layer Security (TLS), only the server authenticates to the client by presenting its certificate. With mutual Transport Layer Security (mTLS), both the server and the client present their certificates so that they are mutually authenticated.

The SSL_CLIENT_AUTHENTICATION parameter controls whether the client certificate needs to be authenticated. This doesn't authenticate or authorize the end user. It authenticates that the certificates used by both the server and client are valid and signed by a known certificate authority (CA). Configuring PKI Certificate Authentication goes into detail about end-user authentication using PKI certificates.

The default for SSL_CLIENT_AUTHENTICATION is TRUE for the database server, listener, and client, which will require mTLS (mutual TLS requiring a client certificate in a client wallet). Settings are as follows:

- OFF/FALSE disables mTLS, which enables one-way TLS.
- ON/TRUE enables mTLS. If it is set to On/TRUE on the server, one-way TLS will be disabled. If it is set to On/TRUE on the client, the client will try to establish mTLS; however, one-way TLS is still allowed if the server is configured with one-way TLS.
- OPTIONAL, server-only configuration value, enables the server to behave as follows:
  - If the client sends a certificate, the connection will be completed as an mTLS connection after the client certificate is authenticated.
  - If the client does not send a certificate, then the connection will be completed as a one-way TLS connection.
- Server Certificate DN Matching
  Oracle recommends using Server certificate DN matching, similar to using server DN matching with one-way TLS, to ensure the client is connecting to the intended server.

Create the Server and Listener Wallet

To get a certificate signed by a publicly signed certificate authority, you must create the database server and listener wallet and export a certificate signing request (CSR).

1. Login to the host where the database is installed.
2. Create the wallet.

```
orapki wallet create -wallet <wallet location> -pwd <wallet password> -
auto_login
```

3. Add the trusted root certificate to the wallet (get this from your certificate administrator).

   ```
   orapki wallet add -wallet <wallet location> -trusted_cert -cert <trusted
   root certificate location>/rootCA.crt -pwd <wallet password>
   ```

4. Create a private key and certificate request in the wallet.

   ```
   orapki wallet add -wallet <wallet location> -keysize 2048 -dn
   <certificate_dn> -pwd <wallet password>
   ```

5. Export the certificate request to get it signed.

   ```
   orapki wallet export -wallet <wallet location> -dn <certificate_dn> -
   request <certificate signing request location>/<file_name>.csr -pwd
   <wallet password>
   ```

6. Display the contents of the wallet.

   ```
   orapki wallet display -wallet <wallet_location>
   ```

   There will be an entry under Requested Certificates.

7. View the contents of the CSR (certificate signing request) file.

   ```
   cat <certificate_signing_request_location>/<file_name>.csr
   ```

8. Send the CSR file to your certificate administrator to have it signed by the root certificate authority (CA) or an intermediate CA.
9. Import the signed database server user certificate into the database wallet.

   ```
   orapki wallet add -wallet <wallet location> -user_cert -cert <signed
   certificate location>/<file_name_signed>.crt -pwd <wallet password>
   ```

10. Display the contents of the wallet:

    ```
    orapki wallet display -wallet <wallet location>
    ```

11. Ensure that the database server user certificate is now displayed under `User Certificates`.

    The wallet you will use for the database server and listener is now ready to be deployed for use.

Set `WALLET_ROOT` and deploy the database server wallet

1. Check to see if `WALLET_ROOT` already exists. Login as a user with privileges to check system parameters and run:

   ```
   SHOW PARAMETER WALLET_ROOT
   ```

   If `WALLET_ROOT` is not already setup, run the next command to create `WALLET_ROOT`.

2. Create `WALLET_ROOT`, a system parameter. Run the following SQL command:

   ```
   alter system set wallet_root = '<wallet_root_directory>' scope=spfile;
   ```

3. Reboot the database.
4. Show the modified `wallet_root` parameter. Run the following SQL command:

   ```
   show parameter wallet_root;
   ```

5. If the TLS directory does not yet exist under `WALLET_ROOT`, create a directory for TLS under your `WALLET_ROOT` PDB directory in the operating system.

   ```
   mkdir -p -v <wallet_root_directory>/<PDB GUID>/tls
   ```

   You can find the PDB GUID for your PDB by running the following SQL command:

   ```
   select guid from v\$containers;
   ```

6. Change ownership of the directory.

   ```
   sudo chown oracle:oinstall -R -v <wallet_root_directory>/<PDB GUID>/tls
   ```

7. Copy the database server ewallet.p12 and the cwallet.sso files to this new tls directory.

   Perform this command from the same directory where the wallets were created:

   ```
   cp ./ewallet.p12 ./cwallet.sso <wallet_root_directory>/<PDB GUID>/tls
   ```

Database server configuration for mTLS

1. Log in to the server where the Oracle database resides.
2. Check that `SSL_CLIENT_AUTHENTICATION` in the `sqlnet.ora` file is set to `TRUE` as this enables mTLS:

   By default, the `sqlnet.ora` file is located in the `$ORACLE_HOME/network/admin` directory or in the location set by the `TNS_ADMIN` environment variable.

   ```
   SSL_CLIENT_AUTHENTICATION=TRUE
   ```

   You may set this to `OPTIONAL` instead which enables both TLS and mTLS and is dependent on whether the client sends the client user certificate.

Listener configuration for mTLS

1. Check the `PROTOCOL` parameter in the `listener.ora` file to ensure TLS is specified.

   By default, `listener.ora` is located in the `$ORACLE_HOME/network/admin` directory.

   The parameter `PROTOCOL=tcps` tells the listener to only use TLS (or mTLS) for database connections.

For example:

```
LISTENER = (ADDRESS=(PROTOCOL=tcps)(HOST=<host_name>)(PORT=1522))
```

2. Ensure that the listener wallet exists in the location of the `WALLET_LOCATION` parameter in the `listener.ora` file. Use the same wallet as you did for the database server.

```
WALLET_LOCATION=
    (SOURCE=
        (METHOD=file)
        (METHOD_DATA=
            (DIRECTORY=$WALLET_DIR/<pdb guid>/tls)))
```

If the listener is on the same server as the database server and the server TLS wallet is in the default location, set the listener `WALLET_LOCATION` to the same location. Alternatively, the server wallet can be copied to a different location for the listener.

If you set the `SSL_SERVER_DN_MATCH` parameter to `TRUE` for DN matching (partial or full DN match), then the hostname or DN check will happen against both the listener certificate and the server certificate. They don't have to be the same certificate, but matching will be done with both certificates.

3. Ensure the `SSL_CLIENT_AUTHENTICATION` parameter is set to `TRUE` in `listener.ora` file to enable mutual TLS.

```
SSL_CLIENT_AUTHENTICATION=TRUE
```

Client Configuration for mTLS

1. Log in to the client for the Oracle database.
2. Set `SSL_CLIENT_AUTHENTICATION` in the `sqlnet.ora` and `tnsnames.ora` files to `TRUE`.

   A setting of `TRUE`, will send a client side user certificate to the server. Because this applies to every connection, you can change the `SSL_CLIENT_AUTHENTICATION` parameter in the `tnsnames.ora` connection string using the same parameter setting which will take precedence over the `sqlnet.ora` setting.

```
SSL_CLIENT_AUTHENTICATION=TRUE
```

> **Tip:**
>
> While the default value for this parameter is true, setting it explicitly to true will make troubleshooting connection problems easier.

3. If you connect to multiple databases and some require mTLS and the other TLS connections don't need a wallet, then you have two options for setting different connections depending if you have a common wallet to connect with the different databases or if each mTLS connection requires a different wallet:

- With a Common Client Wallet

- Without a Common Client Wallet

With a Common Client Wallet

a. Specify a common mTLS client wallet by setting `WALLET_LOCATION` in `sqlnet.ora`. This will result in every mTLS connection using the same client wallet to connect with their database.

b. In the connection string for one-way TLS connections,

   i. Set `SSL_CLIENT_AUTHENTICATION = FALSE` to override the mTLS client wallet setting.

   ii. Set `WALLET_LOCATION = SYSTEM` to specify the system default certificate store.

Without a Common Client Wallet

This can be used if you need to use a different client wallet for each database connection.

a. Set `WALLET_LOCATION = SYSTEM` in `sqlnet.ora` to allow the TLS connections to connect without using a wallet.

b. Set the `WALLET_LOCATION` for every mTLS connection to specify the unique wallet location for each connection.

Related Topics

- Oracle Wallet Location
  Certificates used for TLS are stored in the Oracle wallet. There are several default locations where the wallet can be placed. The location of the wallet can also be configured with the wallet location parameters on the client and listener. The `WALLET_ROOT` system parameter should be used for the database server wallet location.

Connect to the database

Connect to the database using the connection name with the tcps protocol.

```
sqlplus <user_name>@<PDB_name>
```

21.3.2.1 Server Certificate DN Matching

Oracle recommends using Server certificate DN matching, similar to using server DN matching with one-way TLS, to ensure the client is connecting to the intended server.

Configure full DN matching by setting the `SSL_SERVER_CERT_DN` parameter connection string in the `tnsnames.ora` file:

> **Note:**
>
> If you can't set the host value in `tnsnames.ora` or `sqlnet.ora` to the value of the certificate common name (CN) or one of the entries in the SAN field, then consider using full DN matching.
>
> Both the listener and server certificate will be checked with both partial and full DN matching. When using full DN matching, while the server and listener certificate can be different, their DN must be the same for the connection to succeed.

The `tnsnames.ora` file will look similar to:

```
finance=
(DESCRIPTION=
    (ADDRESS_LIST=
        (ADDRESS=(PROTOCOL = tcps)(HOST = finance)
        (PORT = 1575)))
    (CONNECT_DATA=
        (SERVICE_NAME= finance.us.example.com))
    (SECURITY=
        (SSL_SERVER_DN_MATCH = TRUE)
        (SSL_SERVER_CERT_DN="cn=finance,cn=OracleContext,c=us,o=example")))
```

21.3.3 Oracle Wallet Location

Certificates used for TLS are stored in the Oracle wallet. There are several default locations where the wallet can be placed. The location of the wallet can also be configured with the wallet location parameters on the client and listener. The `WALLET_ROOT` system parameter should be used for the database server wallet location.

- Configuring Wallet Location for the Client
  The client's wallet location can be configured using the parameter `WALLET_LOCATION`. When the `WALLET_LOCATION` parameter is configured in `sqlnet.ora`, it applies to all connections. If a connection-specific wallet is needed, `WALLET_LOCATION` for the connection can be configured in the connect string, which overrides `WALLET_LOCATION` in `sqlnet.ora`.
- Configuring Wallet Location for the Listener
  Wallet location for the listener can be configured using the `WALLET_LOCATION` parameter in `listener.ora`.
- Configuring PDB Wallet Location for server
  The multi-tenant architecture enables an Oracle database to function as a multi-tenant container database (CDB) that includes zero, one, or many customer-created pluggable databases (PDBs).
- Oracle Wallet Search Order
  Oracle Database provides several routes for finding the wallet on a server in a Transport Layer Security (TLS) environment.

21.3.3.1 Configuring Wallet Location for the Client

The client's wallet location can be configured using the parameter `WALLET_LOCATION`. When the `WALLET_LOCATION` parameter is configured in `sqlnet.ora`, it applies to all connections. If a

connection-specific wallet is needed, `WALLET_LOCATION` for the connection can be configured in the connect string, which overrides `WALLET_LOCATION` in `sqlnet.ora`.

On certain platforms, a wallet is not required when setting up a client for one-way TLS authentication, and the wallet location is not required in the configuration. Oracle Database can utilize Trusted CA certificates installed on the system to support one-way TLS. Refer to the earlier topic, “Transport Layer Security Connections without a Client Wallet,” for more details and a list of supported platforms.

Static configuration example `(sqlnet.ora)`

```
WALLET_LOCATION =
(SOURCE=
  (METHOD=File)
  (METHOD_DATA=
     (DIRECTORY=your_wallet_dir)
      )
)
```

Dynamic (pre-connection) configuration example `(tnsnames.ora)`

```
svc_name=(DESCRIPTION=
            (ADDRESS=(...))
            (CONNECT_DATA=(...))
            (SECURITY=
                (WALLET_LOCATION=your_wallet_dir)
            )
          )
```

21.3.3.2 Configuring Wallet Location for the Listener

Wallet location for the listener can be configured using the `WALLET_LOCATION` parameter in `listener.ora`.

`WALLET_LOCATION` can be specified for each listener in `listener.ora`.

For example,

```
LISTENER =
    (DESCRIPTION=
        (ADDRESS=
            (PROTOCOL=tcps)
            (HOST=)
            (PORT=5678))
        (SECURITY=
            (WALLET_LOCATION=dir1)))

LISTENER2 =
    (DESCRIPTION=
        (ADDRESS=
            (PROTOCOL=tcps)
            (HOST=)
            (PORT=5679))
        (SECURITY=
            (WALLET_LOCATION=dir2)))
```

21.3.3.3 Configuring PDB Wallet Location for server

The multi-tenant architecture enables an Oracle database to function as a multi-tenant container database (CDB) that includes zero, one, or many customer-created pluggable databases (PDBs).

CDB$ROOT and each PDB can have its own local wallet which can be configured with the `WALLET_ROOT` system parameter defined in the `init.ora` file.

For example, for the CDB root container (this does not appply to all containers in the CDB):

```
WALLET_ROOT/tls
```

For example, for the PDB:

```
WALLET_ROOT/<pdb_GUID>/tls
```

> **Note:**
>
> The parameter `WALLET_LOCATION` is deprecated for use with Oracle Database 23ai for the Oracle Database server. It is not deprecated for use with the Oracle Database client or listener.
> For Oracle Database server, Oracle recommends that you use the `WALLET_ROOT` system parameter instead of using `WALLET_LOCATION`.

21.3.3.4 Oracle Wallet Search Order

Oracle Database provides several routes for finding the wallet on a server in a Transport Layer Security (TLS) environment.

How the Oracle Database Server locates wallets for use in TLS

The Oracle Database server locates the wallet by searching in the following locations in the specified order. If the database has one or more pluggable databases (PDB), the value for `pdb_GUID` must be replaced with the global identifier (GUID) of the PDB.

1. Location defined by the `WALLET_ROOT` system parameter in the `init.ora` file:
   - `WALLET_ROOT/<pdb_ID>/tls` for PDB
   - `WALLET_ROOT/tls` for the CDB root container, `CDB$ROOT`
2. Location defined by the `WALLET_LOCATION` in the `sqlnet.ora` file:
   - `WALLET_LOCATION`

> **Note:**
>
> The parameter `WALLET_LOCATION` is deprecated for use with Oracle Database 23ai for the Oracle Database server. It is not deprecated for use with the Oracle Database client or listener.
> For Oracle Database server, Oracle recommends that you use the `WALLET_ROOT` system parameter instead of using `WALLET_LOCATION`.

3. Location defined by the `$TNS_ADMIN` environment variable. This is the only directory location that will be checked, not any sub-directory underneath this location.
4. Default wallet location:
   - Linux: `/etc/ORACLE/WALLETS/user_name`
     This is the only directory location that will be checked, not any sub-directory underneath this location.
   - Windows: `C:\Users\user_name\ORACLE\WALLETS`
     This is the only directory location that will be checked, not any sub-directory underneath this location.
5. If a single wallet is needed for some or all of the CDB container databases, then place the wallet in `TNS_ADMIN` or the default wallet location. Then the PDB will default to that location when it can't find a wallet under `WALLET_ROOT`.

How the Oracle Database Listener locates wallets for use in TLS

The Oracle Database Listener locates the wallet location by searching in these locations, in the specified order:

1. Location defined by the `WALLET_LOCATION` parameter in the `listener.ora` file
2. Location defined by the `$TNS_ADMIN` environment variable
3. Default wallet location:
   - Linux: `/etc/ORACLE/WALLETS/user_name`
   - Windows: `C:\Users\user_name\ORACLE\WALLETS`

How the Oracle Database Client locates wallets for use in TLS

Oracle Database Client locates the wallet by searching in these locations, in the specified order:

1. Location defined by the `WALLET_LOCATION` parameter in the connection string
2. Location defined by the `WALLET_LOCATION` parameter in the `sqlnet.ora` file
3. Location defined by the `$TNS_ADMIN` environment variable
4. Default wallet location:
   - Linux: `/etc/ORACLE/WALLETS/user_name`
   - Windows: `C:\Users\user_name\ORACLE\WALLETS`
5. System certificate store
   When one-way TLS authentication is desired, Oracle Database Client can use the trusted CA certificates present in the system certificate store. If the client has a need to support client authentication on the connections, it must setup a wallet containing its own certificate along with required trusted CA certificates.

The default certificate store location depends on the platform. At present, Oracle Database supports this method natively on Microsoft Windows and Linux.

For Windows, it is in the Microsoft Certificate Store for Microsoft Windows.

For Linux, its locations are as follows:

- RHEL/Oracle Linux: `/etc/pki/tls/cert.pem`
- Debian/Ubuntu/Gentoo: `/etc/ssl/certs/ca-certificates.crt`
- Fedora/RHEL: `/etc/pki/tls/certs/ca-bundle.crt`
- OpenSUSE: `/etc/ssl/ca-bundle.pem`
- OpenELEC: `/etc/pki/tls/cacert.pem`
- CentOS/RHEL7: `/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem`
- Alpine Linux: `/etc/ssl/cert.pem`

For non-Linux and non-Windows systems, if the PEM file is not in one of the locations listed above for Linux systems, then you must either copy the PEM file to one of these default Linux locations or create a symlink from the PEM file to one of these locations. The file must be a PEM file.

> **Note:**
>
> You cannot change the default location of the certificate store.

21.3.4 Enable Weak DN Matching

The `SSL_ALLOW_WEAK_DN_MATCH` parameter control reverts the DN matching behavior to prior database versions.

Starting in Oracle Database 23ai, the behavior of the `SSL_SERVER_DN_MATCH` parameter has changed.

Server-side certificate verification through distinguished name (DN) is changed as follows: Both the listener certificate and the database server certificate are checked. In earlier Oracle Database releases, only the database server certificate was checked. In most production cases, the same certificate is used by the listener and the database. In cases where different certificates are used, DN matching can require new certificates to allow partial DN matching on SAN or hostname certificate information. In addition to checking the listener certificate, when using partial DN matching is used, the `SERVICE_NAME` parameter will be ignored Only the hostname connect string parameter will be checked against the certificate common name (CN) and subject alternate name (SAN) fields. To revert to the behavior in earlier releases (using the service name in addition to hostname, and only checking the database server certificate), set the new parameter: `SSL_ALLOW_WEAK_DN_MATCH=TRUE`. The default is `FALSE`.

You can set `SSL_ALLOW_WEAK_DN_MATCH` as follows:

- `TRUE` enables `SSL_SERVER_DN_MATCH` to check the database server certificate (but not the listener) and enable the service name to be used for partial DN matching. The search order on the client side is as follows: first, the client sqlnet.ora or connect string host name value is compared against the certificate CN, then the list of names in the subject alternative name (SAN) field. Then the client sqlnet.ora or connect string service_name value is compared against the CN and the list of names in the SAN.

- `FALSE` (the default) disables `SSL_SERVER_DN_MATCH` from checking service name matching. Instead, matching on the client side is based on a search for the `HOST` value in the certificate DN, and if that is not available, then in the subject alternative name (SAN) field (but not the service name). The DN check is performed on the listener and the server certificates.

If you used the service name for partial DN matching previously, then you must either get a new certificate or set `SSL_ALLOW_WEAK_DN_MATCH` to `TRUE` to revert to the pre-release 23ai behavior. You are most likely using the same certificate for both the database server and listener, but if you are not, then you will either need to do one of the following:

- Get a new certificate (use the `orapki wallet add` command for self-signed certificates).
- Change or remove the DN matching strategy.
- Set the `SSL_ALLOW_WEAK_DN_MATCH` parameter to `TRUE` to revert `SSL_SERVER_DN_MATCH` to its older behavior.

When you set `SSL_ALLOW_WEAK_DN_MATCH` to `TRUE`, note the following:

- When the client performs a full DN match (`SSL_SERVER_MATCH=TRUE`, `SSL_SERVER_CERT_DN="certificate_DN"`), then only the database server certificate DN will need to match the `SSL_SERVER_CERT_DN` value.
- When the client performs a partial DN match (`SSL_SERVER_MATCH=TRUE`, `SSL_SERVER_CERT_DN` is not set), then Oracle Database will compare the connect string parameter `HOST` to the common name (CN) of the database server certificate DN and the certificate subject alternate names field (SAN). If there is no partial match, then Oracle Database will continue and check the `SERVICE_NAME` parameter with the CN.

21.3.5 Private Key/Certificate Selection

You can have multiple private key/certificate pairs stored in either the Oracle wallet or a system certificate store to use for certificates. This is sometimes necessary when different databases will assign different client credentials for mTLS, such as for Autonomous Database.

You can only specify the private key/certificate to be used with Windows MCS and Oracle Wallets.

You will need to specify the correct private key/certificate to use for a database connection. By setting the certificate selection parameters for client authentication on Windows, the MSCAPI certificate selection box will not appear, and the matching certificate is automatically used for Transport Layer Security.

While it is more likely that the client will need to select a specific private key/certificate from MCS or the wallet, the server and listener may also need to select a specific certificate for use if there is more than one in the wallet.

- Setting the SSL_CERTIFICATE_ALIAS Parameter
  You can use the `SSL_CERTIFICATE_ALIAS` parameter to specify the alias of the client certificate.
- Setting the SSL_CERTIFICATE_THUMBPRINT Parameter
  You can use the `SSL_CERTIFICATE_THUMBPRINT` to specify the thumbprint of the client certificate.
- Setting the SSL_EXTENDED_KEY_USAGE Parameter
  You can use the `SSL_EXTENDED_KEY_USAGE` parameter to specify the extended key usage of the client certificate.

21.3.5.1 Setting the SSL_CERTIFICATE_ALIAS Parameter

You can use the `SSL_CERTIFICATE_ALIAS` parameter to specify the alias of the client certificate.

1. To get the alias name value, run the following `orapki` command:

   ```
   orapki wallet display -wallet wallet_directory -pwd wallet_password -complete
   ```

   The output will look similar to the following. See the `Alias` field.

   ```
   User Certificates:
   Alias: sslClient
   Subject: CN=ssl
   ClientIssuer: CN=sslRoot,C=US
   Not Before: Thu Jul 18 22:29:17 UTC 2024
   Not After: Sun Jul 16 22:29:17 UTC 2034
   Serial Number: 01
   Key Length: 2048
   MD5 digest: 06:DD:79:AF:E6:D6:70:CE:C3:98:DE:8C:D7:FC:7E:C2
   SHA-256 digest:
   09:B2:EC:FE:A1:B8:C3:F3:F5:A7:DC:C6:00:26:86:BE:39:54:16:93:B6:A8:42:CC:69:
   24:0F:B5:59:43:3F:AA
   SHA-1 digest: 51:25:6F:45:F8:64:E5:CB:9E:56:D2:F2:0C:5C:A6:D5:61:DA:DB:FE
   ```

2. Set the `Alias` value using the `SSL_CERTIFICATE_ALIAS` parameter.
   For example, for `tnsnames.ora`:

   ```
   net_service_name=
         (DESCRIPTION=
           (ADDRESS=(PROTOCOL=tcps)(HOST=sales-svr)(PORT=1522))
           (SECURITY=(SSL_CERTIFICATE_ALIAS=sslClient))
         )
   ```

   This example shows how to set `SSL_CERTIFICATE_ALIAS` in the `sqlnet.ora` file:

   ```
   SSL_CERTIFICATE_ALIAS=sslClient
   ```

Related Topics

- *Oracle Database Net Services Reference*

21.3.5.2 Setting the SSL_CERTIFICATE_THUMBPRINT Parameter

You can use the `SSL_CERTIFICATE_THUMBPRINT` to specify the thumbprint of the client certificate.

The value of the parameter is the SHA-1 or SHA-256 thumbprint of the client certificate, in the *`algorithm`*`:`*`hash`* format

1. To get the thumbprint value, run the following `orapki` command:

   ```
   orapki wallet display -wallet wallet_directory -pwd wallet_password -complete
   ```

The output will look similar to the following. See the `SHA-1 digest` or `SHA-256 digest` field for the thumbprint value.

```
User Certificates:
Alias: sslClient
Subject: CN=ssl
ClientIssuer: CN=sslRoot,C=US
Not Before: Thu Jul 18 22:29:17 UTC 2024
Not After: Sun Jul 16 22:29:17 UTC 2034
Serial Number: 01
Key Length: 2048
MD5 digest: 06:DD:79:AF:E6:D6:70:CE:C3:98:DE:8C:D7:FC:7E:C2
SHA-256 digest:
09:B2:EC:FE:A1:B8:C3:F3:F5:A7:DC:C6:00:26:86:BE:39:54:16:93:B6:A8:42:CC:69:
24:0F:B5:59:43:3F:AA
SHA-1 digest: 51:25:6F:45:F8:64:E5:CB:9E:56:D2:F2:0C:5C:A6:D5:61:DA:DB:FE
```

2. Set this value using the `SSL_CERTIFICATE_THUMBPRINT` parameter. The following example shows how to set it in the `tnsnames.ora` file.
   For example, in the `tnsname.ora` file:

```
net_service_name=
      (DESCRIPTION=
        (ADDRESS=(PROTOCOL=tcps)(HOST=sales-svr)(PORT=1522))

(SECURITY=(SSL_CERTIFICATE_THUMBPRINT=SHA1:1B:11:01:5A:B1:2C:20:B2:12:34:3E
:04:7B:83:47:DE:70:2E:4E:11))
      )
```

   This example shows how to set `SSL_CERTIFICATE_THUMBPRINT` in the `sqlnet.ora` file:

```
SSL_CERTIFICATE_THUMBPRINT=SHA256:B38A5B1A036383922B5DE15361EE03940A56B4564
17E4124419B88EBC61E1123
```

Related Topics

- *Oracle Database Net Services Reference*

21.3.5.3 Setting the SSL_EXTENDED_KEY_USAGE Parameter

You can use the `SSL_EXTENDED_KEY_USAGE` parameter to specify the extended key usage of the client certificate.

You should set the `SQLNET.SSL_EXTENDED_KEY_USAGE` parameter if you have multiple certificates in the security module, but there is only one certificate with extended key usage field of client authentication, and this certificate is the one you want to use to authenticate to the database.

- For example, in the `sqlnet.ora` file:

```
SSL_EXTENDED_KEY_USAGE = "client authentication"
```

You can find the Extended Key Usage from the certificate using `orapki`:

```
orapki cert display -cert <certificate> -complete
```

Related Topics

- *Oracle Database Net Services Reference*

21.3.6 Transport Layer Security Encryption Combined with Authentication Methods

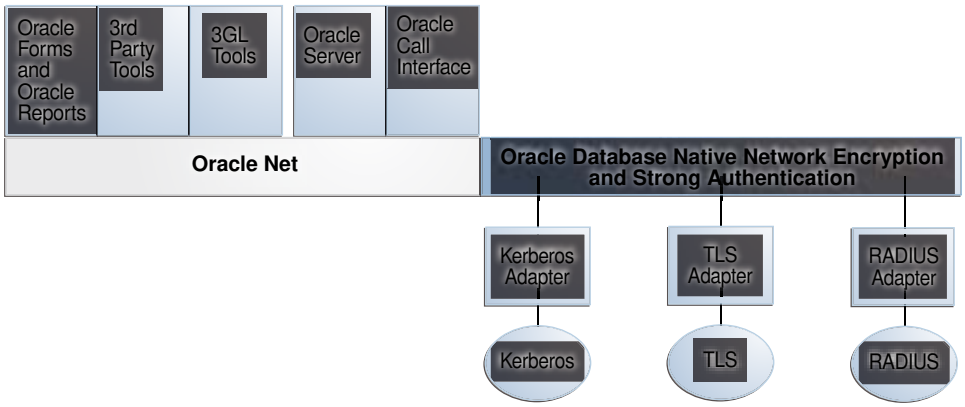
You can configure Oracle Database to use TLS concurrently with all authentication mechanisms such as database user names and passwords, RADIUS, Kerberos, PKI certificates, MS-EI, and OCI IAM.

Architecture: Oracle Database and Transport Layer Security

The Oracle Net Services with Authentication Adapters diagram, displays the Oracle Database implementation of Transport Layer Security architecture, shows that Oracle Databases operates at the session layer on top of TLS and uses TCP/IP at the transport layer. The session layer is a network layer that provides the services needed by the presentation layer entities that enable them to organize and synchronize their dialogue and manage their data exchange. This layer establishes, manages, and terminates network sessions between the client and server. The transport layer is a networking layer that maintains end-to-end reliability through data flow control and error recovery methods.

This separation of functionality lets you employ TLS concurrently with other supported protocols.

Figure 21-1 Oracle Net Services with Authentication Adapters

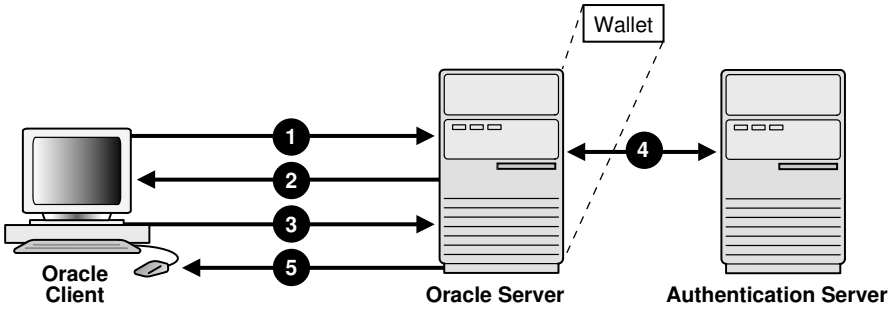


How Transport Layer Security Works with Other Authentication Methods

Transport Layer Security can be used with other authentication methods that Oracle Database supports.

#unique_1275/unique_1275_Connect_42_CIHHEJJB illustrates a configuration in which Transport Layer Security is used in combination with another authentication method.

Figure 21-2 Transport Layer Security in Relation to Other Authentication Methods



In this example, Transport Layer Security is used to establish an encrypted network connection between the client and server, and an alternative authentication method is used to authenticate the client into the database. The process is as follows:

1. The client seeks to connect to the Oracle database server.
2. Transport Layer Security performs a handshake during which the server authenticates itself to the client and both the client and server establish which cipher suite to use.
3. Once the Transport Layer Security handshake is successfully completed, the user seeks access to the database.
4. The Oracle database server authenticates the user with the authentication server using a non-TLS authentication method such as a password, Kerberos, RADIUS, or a cloud identity token (Oracle Cloud Infrastructure (OCI) Identity and Access Management (IAM), Microsoft Azure AD).
5. Upon validation by the authentication method, the Oracle database server grants access and authorization to the user, and then the user can access the database securely by using TLS.

Related Topics

- *Oracle Database Net Services Administrator's Guide*

21.3.7 Specifying TLS Protocol and TLS Cipher Suites

Oracle Database 23ai supports TLS protocol versions 1.2 and 1.3 and their associated cipher suites for Transport Layer Security (TLS).

Oracle provides the configuration parameters `SSL_VERSION`, and `SSL_CIPHER_SUITE` to configure the specific protocol version and cipher suites. However, Oracle recommends that you do not specify these parameters unless required. Omitting these values facilitate auto-detection of the strongest common TLS version (which ensures that the highest available version is selected) and their associated cipher suites. Oracle Database uses the `TLS_AES_256_GCM_SHA384` cipher suite as the default.

- Configuring TLS Protocol Versions
  The `SSL_VERSION` and `TLS_DISABLE_VERSION` parameters define the protocol version of TLS that is enforced at the end point of the component where they are specified.
- Configuring TLS Cipher Suites
  A cipher suite is a set of authentication, encryption, and data integrity algorithms used for exchanging messages between network entities.

- Allowing Certificates from Earlier Algorithms
  You can use certificates that were associated with earlier deprecated (and weaker) algorithms by setting the `ALLOWED_WEAK_CERT_ALGORITHMS` `sqlnet.ora` or `listener.ora` parameter.

21.3.7.1 Configuring TLS Protocol Versions

The `SSL_VERSION` and `TLS_DISABLE_VERSION` parameters define the protocol version of TLS that is enforced at the end point of the component where they are specified.

`SSL_VERSION` and `TLS_DISABLE_VERSION` can be specified with the database server, the listener, the client, or a combination of these components. If the TLS protocol version is specified in more than one of these locations, then at least one version must be common across all of the components. Otherwise, the connection will be rejected. Also, if a TLS protocol version is specified that is not supported by another component, then the connection request will also be rejected.

You can set the `SSL_VERSION` and `TLS_DISABLE_VERSION` parameters in the client or server `sqlnet.ora` or the `listener.ora` file.

`SSL_VERSION` Parameter

In the server `sqlnet.ora` file, set the `SSL_VERSION` parameter to indicate the supported TLS versions on the server.

Valid values are `undetermined` (the default), `TLSv1.2`, and `TLSv1.3`. Separate multiple entries with a comma. For example:

```
SSL_VERSION=(TLSv1.2,TLSv1.3)
```

Append a `+` to the values to specify the minimum version. For example:

```
SSL_VERSION=TLSv1.2+
```

Will allow TLS1.2 and above.

If `SSL_VERSION` and `TLS_DISABLE_VERSION` are not set, or `SSL_VERSION` set to `undetermined`, all supported TLS versions are enabled.

> **Tip:**
>
> Oracle recommends that you do not specify these parameters so that the highest version is auto-detected between server and client.

For environments where you want to enforce TLSv1.3 explicitly, you may specify the protocol version as follows:

```
SSL_VERSION = TLSv1.3
```

`TLS_DISABLE_VERSION` Parameter

In the server `sqlnet.ora` file, set the `TLS_DISABLE_VERSION` parameter to indicate the TLS versions to not allow on the server.

Valid values are `TLSv1.2` and `TLSv1.3`. Separate multiple entries with a comma. For example:

```
TLS_DISABLE_VERSION=(TLSv1.2,TLSv1.3)
```

Will not allow TLS1.2 and TLS1.3.

21.3.7.2 Configuring TLS Cipher Suites

A cipher suite is a set of authentication, encryption, and data integrity algorithms used for exchanging messages between network entities.

During a Transport Layer Security handshake, two entities negotiate to select a cipher suite they will use when transmitting messages to each other through the network.

When you install Oracle Database, the Transport Layer Security cipher suites are set for you by default and negotiated in the order they are listed from the strongest cipher suite. You can override the default order by setting the `SSL_CIPHER_SUITES` parameter. Ensure that you enclose the `SSL_CIPHER_SUITES` parameter setting in parentheses (for example, `SSL_CIPHER_SUITES=(TLS_AES_256_GCM_SHA384)`). Otherwise, the cipher suite setting will not parse correctly.

You can prioritize the cipher suites. When the client negotiates with servers to determine which cipher suite to use, it follows the prioritization you set. When you prioritize the cipher suites, consider the following:

- Compatibility: The server and client must be configured to use compatible cipher suites for a successful connection.
- Cipher priority and strength: Prioritize cipher suites starting with the strongest and moving to the weakest to ensure the highest level of security possible.
- The level of security you want to use: Use this to prevent older clients with weaker cipher suites from connecting to the database
- Strong TLS Cipher Suites
  Oracle provides strong Transport Layer Security (TLS) cipher suites by default. Starting with Oracle Database 23ai, however, Oracle only supports TLSv1.2 and above. The default ciphers supported by Oracle are shown in the table below.
- Deprecated TLS Cipher Suites
  To accommodate legacy products, Oracle provides TLS cipher suites which are considered less secure. Those ciphers are deprecated and disabled by default. The deprecated ciphers supported by Oracle are shown in the table below.
- Enabling Weak Cipher Suites
  You can enable deprecated cipher suites by setting the `SSL_ENABLE_WEAK_CIPHERS` parameter. For the connections to be successful with the weak cipher suites, all three components (client, listener, and server) need to have the weak cipher suites enabled.

21.3.7.2.1 Strong TLS Cipher Suites

Oracle provides strong Transport Layer Security (TLS) cipher suites by default. Starting with Oracle Database 23ai, however, Oracle only supports TLSv1.2 and above. The default ciphers supported by Oracle are shown in the table below.

Table 21-6 Approved TLS Cipher Suites

Cipher Suite	Authentication	Encryption	Data Integrity	TLS Compatibility
TLS_AES_128_CCM_SHA256	ECDHE_RSA, DHE_RSA, ECDHE_ECDSA	AES 128 CCM	SHA256 (SHA 2)	TLS 1.3
TLS_AES_128_GCM_SHA256	ECDHE_RSA, DHE_RSA, ECDHE_ECDSA	AES 128 GCM	SHA256 (SHA-2)	TLS 1.3
TLS_AES_256_GCM_SHA384	ECDHE_RSA, DHE_RSA, ECDHE_ECDSA	AES 256 GCM	SHA384 (SHA-2)	TLS 1.3
TLS_CHACHA20_POLY1305_SHA256 (non-FIPS only)	ECDHE_RSA, DHE_RSA, ECDHE_ECDSA	CHACHA20 POLY1305	SHA256 (SHA-2)	TLS 1.3
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	DHE_RSA	AES 128 GCM	SHA256 (SHA-2)	TLS 1.2
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	DHE_RSA	AES 256 GCM	SHA384 (SHA-2)	TLS 1.2
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDHE_ECDSA	AES 128 GCM	SHA256 (SHA-2)	TLS 1.2
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDHE_ECDSA	AES 256 GCM	SHA384 (SHA-2)	TLS 1.2
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDHE_RSA	AES 128 GCM	SHA256 (SHA-2)	TLS 1.2
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDHE_RSA	AES 256 GCM	SHA384 (SHA-2)	TLS 1.2

21.3.7.2.2 Deprecated TLS Cipher Suites

To accommodate legacy products, Oracle provides TLS cipher suites which are considered less secure. Those ciphers are deprecated and disabled by default. The deprecated ciphers supported by Oracle are shown in the table below.

Table 21-7 Deprecated TLS Cipher Suites

Cipher Suite	Authentication	Encryption	Data Integrity	TLS Compatibility
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	DHE_RSA	AES 128 CBC	SHA256 (SHA-2)	TLS 1.2
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DHE_RSA	AES 256 CBC	SHA (SHA-1)	TLS 1.2
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	DHE_RSA	AES 256 CBC	SHA256 (SHA-2)	TLS 1.2

Table 21-7 (Cont.) Deprecated TLS Cipher Suites

Cipher Suite	Authentication	Encryption	Data Integrity	TLS Compatibility
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	ECDHE_ECDSA	AES 128 CBC	SHA (SHA-1)	TLS 1.2
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	ECDHE_ECDSA	AES 128 CBC	SHA (SHA-1)	TLS 1.2
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	ECDHE_ECDSA	AES 128 CBC	SHA256 (SHA-2)	TLS 1.2
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	ECDHE_ECDSA	AES 256 CBC	SHA (SHA-1)	TLS 1.2
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	ECDHE_ECDSA	AES 256 CBC	SHA384 (SHA-2)	TLS 1.2
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	ECDHE_RSA	AES 128 CBC	SHA (SHA-1)	TLS 1.2
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	ECDHE_RSA	AES 128 CBC	SHA256 (SHA-2)	TLS 1.2
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDHE_RSA	AES 256 CBC	SHA (SHA-1)	TLS 1.2
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	ECDHE_RSA	AES 256 CBC	SHA384 (SHA-2)	TLS 1.2
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	AES 128 CBC	SHA (SHA-1)	TLS 1.2
TLS_RSA_WITH_AES_128_CBC_SHA256	RSA	AES 128 CBC	SHA256 (SHA-2)	TLS 1.2
TLS_RSA_WITH_AES_128_GCM_SHA256	RSA	AES 128 GCM	SHA256 (SHA-2)	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA	RSA	AES 256 CBC	SHA (SHA-1)	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	RSA	AES 256 CBC	SHA256 (SHA-2)	TLS 1.2
TLS_RSA_WITH_AES_256_GCM_SHA384	RSA	AES 256 GCM	SHA384 (SHA-2)	TLS 1.2

21.3.7.2.3 Enabling Weak Cipher Suites

You can enable deprecated cipher suites by setting the `SSL_ENABLE_WEAK_CIPHERS` parameter. For the connections to be successful with the weak cipher suites, all three components (client, listener, and server) need to have the weak cipher suites enabled.

In this specification, `value` can be one of the following:

- `FALSE` (or `OFF`, `NO`, `0`) disables the weak ciphers. The setting is the default. If you try to use a weak cipher, then depending on where you are, the following errors appear:
  - In the database server: `ORA-28860: Fatal SSL error`
  - In the database client: `ORA-29039: There are no matching cipher suites.`
- `TRUE` (or `ON`, `YES`, `1`) enables the weak ciphers.

For example, to enable the deprecated cipher suites,

```
SSL_ENABLE_WEAK_CIPHERS=TRUE
```

21.3.7.3 Allowing Certificates from Earlier Algorithms

You can use certificates that were associated with earlier deprecated (and weaker) algorithms by setting the `ALLOWED_WEAK_CERT_ALGORITHMS` `sqlnet.ora` or `listener.ora` parameter.

The `ALLOWED_WEAK_CERT_ALGORITHMS` allows you to explicitly enable earlier algorithms. However, be aware that earlier algorithms are less secure than newer algorithms. This parameter replaces the `ALLOW_MD5_CERTS` and `ALLOW_SHA1_CERTS` parameters, which are deprecated starting in Oracle Database release 23ai.

1. Log in to the database server or the client server.
2. Edit the `sqlnet.ora` or `listener.ora` parameter file to include the `ALLOWED_WEAK_CERT_ALGORITHMS` parameter.

   `MD5` is disabled by default and `SHA1` is enabled by default. The default location of the `sqlnet.ora` file is in the `$ORACLE_HOME/network/admin` directory.

   You can specify:

   - `SHA1` - enabled by default, enables SHA1 and disables MD5
   - `MD5` - enables MD5 and disables SHA1
   - `NONE` - both MD5 and SHA1 are disabled

   If you want to specify both SHA1 and MD5, then separate them with a comma. For example:

   ```
   ALLOWED_WEAK_CERT_ALGORITHMS = (MD5,SHA1)
   ```

21.3.8 Certificate Validation with Certificate Revocation Lists

Oracle provides tools that enable you to validate certificates using certificate revocation lists.

- About Certificate Validation with Certificate Revocation Lists
  The process of determining whether a given certificate can be used in a given context is referred to as certificate validation.

- What CRLs Should You Use?
  You should have CRLs for all of the trust points that you honor.
- How CRL Checking Works
  Oracle Database checks the certificate revocation status against CRLs.
- Configuring Certificate Validation with Certificate Revocation Lists
  You can edit the `sqlnet.ora` file to configure certificate validation with certificate revocation lists.
- Certificate Revocation List Management
  Certificate revocation list management entails ensuring that the CRLs are the correct format before you enable certificate revocation checking.
- Troubleshooting CRL Certificate Validation
  To determine whether certificates are being validated against CRLs, you can enable Oracle Net tracing.
- Oracle Net Tracing File Error Messages Associated with Certificate Validation
  Oracle generates trace messages that are relevant to certificate validation.

21.3.8.1 About Certificate Validation with Certificate Revocation Lists

The process of determining whether a given certificate can be used in a given context is referred to as certificate validation.

Certificate validation includes determining that the following takes place:

- A trusted certificate authority (CA) has digitally signed the certificate
- The certificate's digital signature corresponds to the independently-calculated hash value of the certificate itself and the certificate signer's (CA's) public key
- The certificate has not expired
- The certificate has not been revoked

The Transport Layer Security network layer automatically performs the first three validation checks, but you must configure certificate revocation list (CRL) checking to ensure that certificates have not been revoked. CRLs are signed data structures that contain a list of revoked certificates. They are usually issued and signed by the same entity who issued the original certificate.

21.3.8.2 What CRLs Should You Use?

You should have CRLs for all of the trust points that you honor.

The trust points are the trusted certificates from a third party identity that is qualified with a level of trust.

Typically, the certificate authorities you trust are called trust points.

21.3.8.3 How CRL Checking Works

Oracle Database checks the certificate revocation status against CRLs.

These CRLs are located in file system directories, Oracle Internet Directory, or downloaded from the location specified in the CRL Distribution Point (CRL DP) extension on the certificate.

Typically, CRL definitions are valid for a few days. If you store your CRLs on the local file system or in the directory, then you must update them regularly. If you use a CRL Distribution

Point (CRL DP), then CRLs are downloaded each time a certificate is used, so there is no need to regularly refresh the CRLs.

The server searches for CRLs in the following locations in the order listed. When the system finds a CRL that matches the certificate CA's DN, it stops searching.

1. Local file system

   The server checks the `sqlnet.ora` file for the `SSL_CRL_FILE` parameter first, followed by the `SSL_CRL_PATH` parameter. If these two parameters are not specified, then the server checks the wallet location for any CRLs.

   > **Note:**
   >
   > If you store CRLs on your local file system, then you must use the `orapki` utility to periodically update them (for example, renaming CRLs with a hash value for certificate validation).

2. Oracle Internet Directory

   If the server cannot locate the CRL on the local file system and directory connection information has been configured in an `ldap.ora` file, then the server searches in the directory. It searches the CRL subtree by using the CA's distinguished name (DN) and the DN of the CRL subtree.

   The server must have a properly configured `ldap.ora` file to search for CRLs in the directory. It cannot use the Domain Name System (DNS) discovery feature of Oracle Internet Directory. Also note that if you store CRLs in the directory, then you must use the `orapki` utility to periodically update them.

3. CRL DP

   If the CA specifies a location in the CRL DP X.509, version 3, certificate extension when the certificate is issued, then the appropriate CRL that contains revocation information for that certificate is downloaded. Currently, Oracle Database supports downloading CRLs over LDAP.

   Note the following:

   - For performance reasons, only user certificates are checked.
   - Oracle recommends that you store CRLs in the directory rather than the local file system.

Related Topics

- Uploading CRLs to Oracle Internet Directory
  Publishing CRLs in the directory enables CRL validation throughout your enterprise, eliminating the need for individual applications to configure their own CRLs.
- Renaming CRLs with a Hash Value for Certificate Validation
  When the system validates a certificate, it must locate the CRL issued by the CA who created the certificate.

21.3.8.4 Configuring Certificate Validation with Certificate Revocation Lists

You can edit the `sqlnet.ora` file to configure certificate validation with certificate revocation lists.

- About Configuring Certificate Validation with Certificate Revocation Lists
  The `SSL_CERT_REVOCATION` parameter must be set to `REQUIRED` or `REQUESTED` in the `sqlnet.ora` file to enable certificate revocation status checking.
- Enabling Certificate Revocation Status Checking for the Client or Server
  You can enable certificate revocation status checking for a client or a server.
- Disabling Certificate Revocation Status Checking
  You can disable certificate revocation status checking.

21.3.8.4.1 About Configuring Certificate Validation with Certificate Revocation Lists

The `SSL_CERT_REVOCATION` parameter must be set to `REQUIRED` or `REQUESTED` in the `sqlnet.ora` file to enable certificate revocation status checking.

By default this parameter is set to `NONE` indicating that certificate revocation status checking is turned off.

> **Note:**
>
> If you want to store CRLs on your local file system or in Oracle Internet Directory, then you must use the command line utility, `orapki`, to rename CRLs in your file system or upload them to the directory.

Related Topics

- Certificate Revocation List Management
  Certificate revocation list management entails ensuring that the CRLs are the correct format before you enable certificate revocation checking.

21.3.8.4.2 Enabling Certificate Revocation Status Checking for the Client or Server

You can enable certificate revocation status checking for a client or a server.

1. Log in to the Oracle Database server.
2. Modify the `SSL_CERT_REVOCATION` parameter in the `sqlnet.ora` file.

   ```
   SSL_CERT_REVOCATION=value
   ```

   In this specification, *`value`* can be either of the following settings:

   - `required` requires certificate revocation status checking. The TLS connection is rejected if a certificate is revoked or no CRL is found. TLS connections are accepted only if it can be verified that the certificate has not been revoked.
   - `requested` performs certificate revocation status checking if a CRL is available. The TLS connection is rejected if a certificate is revoked. TLS connections are accepted if no CRL is found or if the certificate has not been revoked. For performance reasons, only user certificates are checked for revocation.
3. Optionally, modify the `TLS_CERT_VALIDATION_MODE` parameter in the `sqlnet.ora` file.

   ```
   TLS_CERT_VALIDATION=value
   ```

In this specification, *value* can be either of the following settings:

- `strict` would enforce stricter checks on CA certificate validation following RFC 5280.
- `non-strict` is the default value and means that the CA certificate validations would be relaxed and would not follow all constraints of RFC 5280.

4. If CRLs are stored on your local file system, then set one or both of the following `sqlnet.ora` parameters that specify where they are stored.
   - `SSL_CRL_PATH` sets the path to the directory where CRLs are stored. If you omit this setting, then the default is the wallet directory. Both DER-encoded (binary format) and PEM-encoded (BASE64) CRLs are supported. If you want to store CRLs in a local file system directory, then you must use the `orapki` utility to rename them so the system can locate them.
   - `SSL_CRL_FILE` sets the path to a comprehensive CRL file (where PEM-encoded (BASE64) CRLs are concatenated in order of preference in one file). Ensure that the file is present in the specified location, or else the application will not be able to start.
5. If you want to fetch CRLs from Oracle Internet Directory, then edit the `ldap.ora` file to include the directory server and port information.

   When configuring your `ldap.ora` file, you should specify only a non-TLS port for the directory. CRL download is done as part of the TLS protocol, and making a TLS connection within a TLS connection is not supported.

   Oracle Database CRL functionality will not work if the Oracle Internet Directory non-TLS port is disabled.
6. Repeat these steps for the Oracle Database client `sqlnet.ora` file.

Related Topics

- Renaming CRLs with a Hash Value for Certificate Validation
  When the system validates a certificate, it must locate the CRL issued by the CA who created the certificate.

21.3.8.4.3 Disabling Certificate Revocation Status Checking

You can disable certificate revocation status checking.

1. Log in to the Oracle Database server.
2. Modify the `SSL_CERT_REVOCATION` parameter in the `sqlnet.ora` file as follows:

   ```
   SSL_CERT_REVOCATION=NONE
   ```

3. Repeat this step for the Oracle Database client.

Related Topics

- Troubleshooting CRL Certificate Validation
  To determine whether certificates are being validated against CRLs, you can enable Oracle Net tracing.

21.3.8.5 Certificate Revocation List Management

Certificate revocation list management entails ensuring that the CRLs are the correct format before you enable certificate revocation checking.

- About Certificate Revocation List Management
  Oracle Database provides a command-line utility, `orapki`, that you can use to manage certificate revocation lists (CRL).
- Displaying orapki Help for Commands That Manage CRLs
  You can display all the `orapki` commands that are available for managing CRLs.
- Renaming CRLs with a Hash Value for Certificate Validation
  When the system validates a certificate, it must locate the CRL issued by the CA who created the certificate.
- Uploading CRLs to Oracle Internet Directory
  Publishing CRLs in the directory enables CRL validation throughout your enterprise, eliminating the need for individual applications to configure their own CRLs.
- Listing CRLs Stored in Oracle Internet Directory
  You can display a list of all CRLs stored in the directory with `orapki`, which is useful for browsing to locate a particular CRL to view or download to your local computer.
- Viewing CRLs in Oracle Internet Directory
  Oracle Internet Directory CRLs are available in a summarized format; you also can request a listing of revoked certificates for a CRL.
- Deleting CRLs from Oracle Internet Directory
  The user who deletes CRLs from the directory by using `orapki` must be a member of the directory group `CRLAdmins`.

21.3.8.5.1 About Certificate Revocation List Management

Oracle Database provides a command-line utility, `orapki`, that you can use to manage certificate revocation lists (CRL).

Before you can enable certificate revocation status checking, you must ensure that the CRLs you receive from the CAs you use are in a form (renamed with a hash value) or in a location (uploaded to the directory) where your computer can use them.

You can also use LDAP command-line tools to manage CRLs in Oracle Internet Directory.

> **Note:**
>
> CRLs must be updated at regular intervals (before they expire) for successful validation. You can automate this task by using `orapki` commands in a script

21.3.8.5.2 Displaying orapki Help for Commands That Manage CRLs

You can display all the `orapki` commands that are available for managing CRLs.

- To display all the `orapki` available CRL management commands and their options, enter the following at the command line:

```
orapki crl help
```

> **Note:**
>
> Using the `-summary`, `-complete`, or `-wallet` command options is always optional. A command will still run if these command options are not specified.

21.3.8.5.3 Renaming CRLs with a Hash Value for Certificate Validation

When the system validates a certificate, it must locate the CRL issued by the CA who created the certificate.

The system locates the appropriate CRL by matching the issuer name in the certificate with the issuer name in the CRL.

When you specify a CRL storage location for the **Certificate Revocation Lists Path** field in Oracle Net Manager, which sets the `SSL_CRL_PATH` parameter in the `sqlnet.ora` file, use the `orapki` utility to rename CRLs with a hash value that represents the issuer's name. Creating the hash value enables the server to load the CRLs.

On UNIX operating systems, `orapki` creates a symbolic link to the CRL. On Windows operating systems, it creates a copy of the CRL file. In either case, the symbolic link or the copy created by `orapki` are named with a hash value of the issuer's name. Then when the system validates a certificate, the same hash function is used to calculate the link (or copy) name so the appropriate CRL can be loaded.

- Depending on the operating system, enter one of the following commands to rename CRLs stored in the file system:
  - To rename CRLs stored in UNIX file systems:

    ```
    orapki crl hash -crl crl_filename [-wallet wallet_location] -symlink
    crl_directory [-summary]
    ```

  - To rename CRLs stored in Windows file systems:

    ```
    orapki crl hash -crl crl_filename [-wallet wallet_location] -copy crl_directory
    [-summary]
    ```

In this specification, `crl_filename` is the name of the CRL file, `wallet_location` is the location of a wallet that contains the certificate of the CA that issued the CRL, and `crl_directory` is the directory where the CRL is located.

Using `-wallet` and `-summary` are optional. Specifying `-wallet` causes the tool to verify the validity of the CRL against the CA's certificate prior to renaming the CRL. Specifying the `-summary` option causes the tool to display the CRL issuer's name.

21.3.8.5.4 Uploading CRLs to Oracle Internet Directory

Publishing CRLs in the directory enables CRL validation throughout your enterprise, eliminating the need for individual applications to configure their own CRLs.

All applications can use the CRLs stored in the directory where they can be centrally managed, greatly reducing the administrative overhead of CRL management and use. The user who uploads CRLs to the directory by using `orapki` must be a member of the directory group `CRLAdmins` (`cn=CRLAdmins,cn=groups,%s_OracleContextDN%`). This is a privileged operation because these CRLs are accessible to the entire enterprise. Contact your directory administrator to get added to this administrative directory group.

- To upload CRLs to the directory, enter the following at the command line:

```
orapki crl upload -crl crl_location -ldap hostname:ssl_port -user username [-wallet
wallet_location] [-summary]
```

In this specification, *crl_location* is the file name or URL where the CRL is located, *hostname* and *ssl_port* (TLS port with no authentication) are for the system on which your directory is installed, *username* is the directory user who has permission to add CRLs to the CRL subtree, and wallet_location is the location of a wallet that contains the certificate of the CA that issued the CRL.

Using -wallet and -summary are optional. Specifying -wallet causes the tool to verify the validity of the CRL against the CA's certificate prior to uploading it to the directory. Specifying the -summary option causes the tool to print the CRL issuer's name and the LDAP entry where the CRL is stored in the directory.

The following example illustrates uploading a CRL with the orapki utility:

```
orapki crl upload -crl /home/user1/wallet/crldir/crl.txt -ldap host1.example.com:3533 -
user cn=orcladmin
```

Note:

- The orapki utility will prompt you for the directory password when you perform this operation.
- Ensure that you specify the directory SSL port on which the Diffie-Hellman-based TLS server is running. This is the TLS port that does not perform authentication. Neither the server authentication nor the mutual authentication TLS ports are supported by the orapki utility.

21.3.8.5.5 Listing CRLs Stored in Oracle Internet Directory

You can display a list of all CRLs stored in the directory with orapki, which is useful for browsing to locate a particular CRL to view or download to your local computer.

This command displays the CA who issued the CRL (Issuer) and its location (DN) in the CRL subtree of your directory.

- To list CRLs in Oracle Internet Directory, enter the following at the command line:

  ```
  orapki crl list -ldap hostname:ssl_port
  ```

  where the *hostname* and *ssl_port* are for the system on which your directory is installed.

Note:

This is the directory SSL port with no authentication as described in the preceding section. Uploading CRLs to Oracle Internet Directory

21.3.8.5.6 Viewing CRLs in Oracle Internet Directory

Oracle Internet Directory CRLs are available in a summarized format; you also can request a listing of revoked certificates for a CRL.

You can view CRLs stored in Oracle Internet Directory in a summarized format or you can request a complete listing of revoked certificates for a CRL. A summary listing provides the CRL issuer's name and its validity period. A complete listing provides a list of all revoked certificates contained in the CRL.

- To view a summary listing of a CRL in Oracle Internet Directory, enter the following at the command line:

  ```
  orapki crl display -crl crl_location [-wallet wallet_location] -summary
  ```

  In this specification, *crl_location* is the location of the CRL in the directory. It is convenient to paste the CRL location from the list that displays when you use the `orapki crl list` command.

  To view a list of all revoked certificates contained in a specified CRL, which is stored in Oracle Internet Directory, you can enter the following at the command line:

  ```
  orapki crl display -crl crl_location [-wallet wallet_location] -complete
  ```

  For example, the following `orapki` command:

  ```
  orapki crl display -crl $T_WORK/pki/wlt_crl/nzcrl.txt -wallet $T_WORK/pki/wlt_crl -
  complete
  ```

  produces the following output, which lists the CRL issuer's DN, its publication date, date of its next update, and the revoked certificates it contains:

  ```
  issuer = CN=root,C=us, thisUpdate = Sun Nov 16 10:56:58 PST 2003, nextUpdate = Mon
  Sep 30 11:56:58 PDT 2013, revokedCertificates = {(serialNo =
  153328337133459399575438325845117876415, revocationDate - Sun Nov 16 10:56:58 PST
  2003)}
  CRL is valid
  ```

  Using the `-wallet` option causes the `orapki crl display` command to validate the CRL against the CA's certificate.

  Depending on the size of your CRL, choosing the `-complete` option may take a long time to display.

  You can also use Oracle Directory Manager, a graphical user interface tool that is provided with Oracle Internet Directory, to view CRLs in the directory. CRLs are stored in the following directory location:

  ```
  cn=CRLValidation,cn=Validation,cn=PKI,cn=Products,cn=OracleContext
  ```

Related Topics

- Listing CRLs Stored in Oracle Internet Directory
  You can display a list of all CRLs stored in the directory with `orapki`, which is useful for browsing to locate a particular CRL to view or download to your local computer.

21.3.8.5.7 Deleting CRLs from Oracle Internet Directory

The user who deletes CRLs from the directory by using `orapki` must be a member of the directory group `CRLAdmins`.

- To delete CRLs from the directory, enter the following at the command line:

  ```
  orapki crl delete -issuer issuer_name -ldap host:ssl_port -user username [-summary]
  ```

  In this specification, *issuer_name* is the name of the CA who issued the CRL, the `hostname` and *ssl_port* are for the system on which your directory is installed, and `username` is the directory user who has permission to delete CRLs from the CRL subtree. Ensure that this must be a directory SSL port with no authentication.

  Using the `-summary` option causes the tool to print the CRL LDAP entry that was deleted.

  For example, the following `orapki` command:

  ```
  orapki crl delete -issuer "CN=root,C=us" -ldap machine1:3500 -user cn=orcladmin -
  summary
  ```

  produces the following output, which lists the location of the deleted CRL in the directory:

  ```
  Deleted CRL at cn=root
  cd45860c.rN,cn=CRLValidation,cn=Validation,cn=PKI,cn=Products,cn=OracleContext
  ```

Related Topics

- Uploading CRLs to Oracle Internet Directory
  Publishing CRLs in the directory enables CRL validation throughout your enterprise, eliminating the need for individual applications to configure their own CRLs.

21.3.8.6 Troubleshooting CRL Certificate Validation

To determine whether certificates are being validated against CRLs, you can enable Oracle Net tracing.

When a revoked certificate is validated by using CRLs, then you will see the following entries in the Oracle Net tracing file without error messages logged between `entry` and `exit`:

```
nzcrlVCS_VerifyCRLSignature: entry
nzcrlVCS_VerifyCRLSignature: exit

nzcrlVCD_VerifyCRLDate: entry
nzcrlVCD_VerifyCRLDate: exit

nzcrlCCS_CheckCertStatus: entry
nzcrlCCS_CheckCertStatus: Certificate is listed in CRL
nzcrlCCS_CheckCertStatus: exit
```

> **Note:**
>
> Note that when certificate validation fails, the peer in the SSL handshake sees an `ORA-29024: Certificate Validation Failure`.

Related Topics

- Oracle Net Tracing File Error Messages Associated with Certificate Validation
  Oracle generates trace messages that are relevant to certificate validation.
- *Oracle Database Net Services Administrator's Guide*

21.3.8.7 Oracle Net Tracing File Error Messages Associated with Certificate Validation

Oracle generates trace messages that are relevant to certificate validation.

These trace messages may be logged between the `entry` and `exit` entries in the Oracle Net tracing file. Oracle SSL looks for CRLs in multiple locations, so there may be multiple errors in the trace.

You can check the following list of possible error messages for information about how to resolve them.

CRL signature verification failed
Cause: The CRL signature cannot be verified.

Action: Ensure that the downloaded CRL is issued by the peer's CA and that the CRL was not corrupted when it was downloaded. Note that the `orapki` utility verifies the CRL before renaming it with a hash value or before uploading it to the directory.
See Certificate Revocation List Management for information about using `orapki` for CRL management.

CRL date verification failed
Cause: The current time is later than the time listed in the next update field. You should not see this error if CRL DP is used. The system searches for the CRL in the following order:

1. File system
2. Oracle Internet Directory
3. CRL DP

The first CRL found in this search may not be the latest.

Action: Update the CRL with the most recent copy.

CRL could not be found
Cause: The CRL could not be found at the configured locations. This will return error ORA-29024 if the configuration specifies that certificate validation is required.

Action: Ensure that the CRL locations specified in the configuration are correct by performing the following steps:

1. Use Oracle Net Manager to check if the correct CRL location is configured. Refer to Configuring Certificate Validation with Certificate Revocation Lists
2. If necessary, use the `orapki` utility to configure CRLs for system use as follows:
   - For CRLs stored on your local file system, refer to Renaming CRLs with a Hash Value for Certificate Validation
   - CRLs stored in the directory, refer to Uploading CRLs to Oracle Internet Directory

Oracle Internet Directory host name or port number not set
Cause: Oracle Internet Directory connection information is not set. Note that this is not an irrecoverable error. The search continues with CRL DP.

Action: If you want to store the CRLs in Oracle Internet Directory, then use Oracle Net Configuration Assistant to create and configure an `ldap.ora` file for your Oracle home.

Fetch CRL from CRL DP: No CRLs found
Cause: The CRL could not be fetched by using the CRL Distribution Point (CRL DP). This happens if the certificate does not have a location specified in its CRL DP extension, or if the URL specified in the CRL DP extension is incorrect.

Action: Ensure that your certificate authority publishes the CRL to the URL that is specified in the certificate's CRL DP extension.
Manually download the CRL. Then depending on whether you want to store it on your local file system or in Oracle Internet Directory, perform the following steps:
If you want to store the CRL on your local file system:

1. Use Oracle Net Manager to specify the path to the CRL directory or file. Refer to Configuring Certificate Validation with Certificate Revocation Lists
2. Use the `orapki` utility to configure the CRL for system use. Refer to Renaming CRLs with a Hash Value for Certificate Validation

If you want to store the CRL in Oracle Internet Directory:

1. Use Oracle Net Configuration Assistant to create and configure an `ldap.ora` file with directory connection information.
2. Use the `orapki` utility to upload the CRL to the directory. Refer to Uploading CRLs to Oracle Internet Directory

21.4 TLS and Other Oracle Products

Transport Layer Security (TLS) can be configured when using other Oracle Database products.

- Transport Layer Security Connections in an Oracle Real Application Clusters Environment
  You can configure Transport Layer Security (TLS) connections in an Oracle Real Application Clusters (Oracle RAC) environment by using Oracle RAC tools and modifying Oracle Database configuration files.

21.4.1 Transport Layer Security Connections in an Oracle Real Application Clusters Environment

You can configure Transport Layer Security (TLS) connections in an Oracle Real Application Clusters (Oracle RAC) environment by using Oracle RAC tools and modifying Oracle Database configuration files.

- Step 1: Configure TCPS Protocol Endpoints
  In Oracle Real Application Clusters (Oracle RAC), clients access one of three scan listeners and are then routed to database listeners. To support Transport Layer Security (TLS), all of these listeners must have TCPS protocol endpoints.
- Step 2: Ensure That the LOCAL_LISTENER Parameter Is Correctly Set on Each Node
  The Oracle Agent automatically sets the `LOCAL_LISTENER` parameter on each node, but you should double-check to ensure that it is correct.
- Step 3: Create Transport Layer Security Wallets and Certificates
  You must create Transport Layer Security (TLS) wallets and certificates for the cluster and also for clients that will connect to the cluster over TLS.
- Step 4: Create a Wallet in Each Node of the Oracle RAC Cluster
  After you have created the cluster wallet, you can copy it to each node of the Oracle Real Applications (Oracle RAC) cluster.

- Step 5: Define Wallet Locations in the listener.ora and sqlnet.ora Files
  To enable the database server and listeners to access the wallets, you must define the wallet locations in the `listener.ora` and `sqlnet.ora` files.
- Step 6: Restart the Database Instances and Listeners
  With the wallets in place and the `*.ora` files edited, you must restart the database server and listener processes so that they pick up the new settings.
- Step 7: Test the Cluster Node Configuration
  To test the cluster node configuration, you can create a connect descriptor for the node and then try to connect to this node.
- Step 8: Test the Remote Client Configuration
  After you have tested the wallet on the Oracle Real Applications (Oracle RAC) cluster nodes, you are ready to test the remote client configuration.

21.4.1.1 Step 1: Configure TCPS Protocol Endpoints

In Oracle Real Application Clusters (Oracle RAC), clients access one of three scan listeners and are then routed to database listeners. To support Transport Layer Security (TLS), all of these listeners must have TCPS protocol endpoints.

1. Log in to the cluster that hosts the Oracle RAC database.
2. Check the listener resources to find if they support TCP endpoints.

   For example:

   ```
   $ srvctl config listener -h
   ```

   Output similar to the following appears:

   ```
   Name: LISTENER
   Subnet: 192.0.2.195
   Type: type
   Owner: pfitch
   Home: Grid_home
   End points: TCP:1521
   ```

   The following command displays information about the scan listener:

   ```
   $ srvctl config scan_listener -h
   ```

   Output similar to the following appears:

   ```
   SCAN Listener LISTENER_SCAN1 exists. Port: TCP:1529
   Registration invited nodes:
   Registration invited subnets:
   SCAN Listener is enabled.
   SCAN Listener is individually enabled on nodes:
   SCAN Listener is individually disabled on nodes:
   ```

3. Add TCPS endpoints to the database listeners.

For example:

```
$ srvctl modify listener -endpoints "TCP:port_1/TCPS:port_2"
```

4. Check the listener configuration.

   For example:

```
$ srvctl config listener

Name: LISTENER
Network: 1, Owner: oracle
Home: CRS_home
End points: TCP:port_1/TCPS:port_2

$ lsnrctl status

Listening Endpoints Summary...
(DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=LISTENER)))
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcps)(HOST=IP_address)(PORT=port_2)))
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=IP_address)(PORT=port_1)))
```

5. Add TCPS endpoints to the scan listeners.

   For example:

```
$ srvctl modify scan_listener -endpoints "TCP:port_1/TCPS:port_2"
```

6. Check the scan listener configuration.

   For example:

```
$ srvctl config scan_listener

SCAN Listener LISTENER_SCAN1 exists. Port: TCP:port_1/TCPS:port_2
SCAN Listener LISTENER_SCAN2 exists. Port: TCP:port_1/TCPS:port_2
SCAN Listener LISTENER_SCAN3 exists. Port: TCP:port_1/TCPS:port_2

$ lsnrctl status listener_scan3

Listening Endpoints Summary...
(DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=LISTENER_SCAN3)))
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=IP_address)(PORT=port_1)))
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcps)(HOST=IP_address)(PORT=port_2)))
```

21.4.1.2 Step 2: Ensure That the LOCAL_LISTENER Parameter Is Correctly Set on Each Node

The Oracle Agent automatically sets the `LOCAL_LISTENER` parameter on each node, but you should double-check to ensure that it is correct.

1. Log in any Oracle Real Application Clusters (Oracle RAC) node.

2. In SQL*Plus, as a user with the `SYSDBA` administrative privilege, check the `LOCAL_LISTENER` parameter.

```
show parameter local_listener;
```

Output similar to the following appears:

```
NAME                                 TYPE        VALUE
------------------------------------ -----------
------------------------------
local_listener                       string
(DESCRIPTION=(ADDRESS_LIST=
                                                 (ADDRESS=(PROTOCOL=TCPS)
(HOST=IP_address)
                                                 (PORT=port_2))))
```

3. If the output is not what you want, then restart each Oracle RAC instance.

21.4.1.3 Step 3: Create Transport Layer Security Wallets and Certificates

You must create Transport Layer Security (TLS) wallets and certificates for the cluster and also for clients that will connect to the cluster over TLS.

- Oracle Real Application Clusters Components That Need Certificates
  Specific components in Oracle Real Application Clusters (Oracle RAC) need certificates when you configure Transport Layer Security (TLS) connections.
- Creating Transport Layer Security Wallets and Certificates
  To create the Transport Layer Security wallets and certificates, you first need to create the root CA certificate, followed by the cluster and client wallets.

21.4.1.3.1 Oracle Real Application Clusters Components That Need Certificates

Specific components in Oracle Real Application Clusters (Oracle RAC) need certificates when you configure Transport Layer Security (TLS) connections.

- Each cluster node (server) and listener must have a wallet with the user certificate and CA certificates.
- The client only needs CA certificates of the listeners and servers (either in wallet or system's certificate store) if one-way TLS is configured.
- The client needs a wallet with its user certificate and CA certificates of the listeners and servers if mTLS is configured.

21.4.1.3.2 Creating Transport Layer Security Wallets and Certificates

To create the Transport Layer Security wallets and certificates, you first need to create the root CA certificate, followed by the cluster and client wallets.

1. Create the root CA certificate.
   a. Log in to any Oracle Real Application Clusters (Oracle RAC) cluster node.
   b. Use the `orapki` utility to create the CA wallet in a directory for the CA.

```
$ orapki wallet create -wallet <CA__wallet_directory>
```

c. Create a self-signed root certificate for the CA wallet.

```
$ orapki wallet add -wallet <CA__wallet_directory> -self_signed -dn
"CN=test CA,O=test,C=c" -keysize 2048 -validity 3650 -sign_alg sha256
```

d. Extract the root CA certificate from the wallet.

This root certificate will be used as the trusted CA certificate in cluster and client wallets and can be distributed or published for users who are managing the PKCS#12 wallets.

```
$ orapki wallet export -wallet <CA__wallet_directory> -dn "CN=test
CA,O=test,C=c" -cert testCAroot.cer
```

To check the configuration:

```
 $ orapki wallet display -wallet <CA__wallet_directory>
```

Output similar to the following appears:

```
Requested Certificates:
User Certificates:
Subject:        CN=test CA,O=test,C=c
Trusted Certificates:
Subject:        CN=test CA,O=test,C=c
```

2. Create the cluster wallet.

Follow the remaining steps in this procedure to sign the user certificate requests and provide authorized digital user certificates to different entities and processes in your environments. Repeat this process for each entity in the test environment that participates in the public key infrastructure functionality. A valid wallet consists of a root CA certificate and the signed user certificate.

a. Create a wallet that is in a different location from the from the CA home directory.

```
$ orapki wallet create -wallet <cluster_wallet_directory>
```

b. Create a user identity (user `dn`) and then export the certificate request.

```
$ orapki wallet add -wallet <cluster_wallet_directory> -dn
"CN=testuser" -keysize 2048
```

```
$ orapki wallet export -wallet <cluster_wallet_directory> -dn
"CN=testuser" -request <cluster_wallet_directory>/testuser.req
```

At this stage, the *<cluster_wallet_directory>* directory will contain the wallet (`ewallet.p12`) and the certificate request (`testuser.req`). The certificate request can be signed by the CA generated above.

```
$ orapki cert create -wallet <CA__wallet_directory> -request
<CA__wallet_directory>/testuser.req -cert <cluster_wallet_directory>/
testuser.cer -validity 3650 -sign_alg sha256
```

The *<cluster_wallet_directory>* directory now has the `testuser.cer` certificate request file.

c. Import the root certificate (`testCAroot.cer`) and the signed user certificate (`testuser.cer`) into the user wallet.

```
$ orapki wallet add -wallet <cluster_wallet_directory> -trusted_cert -
cert <CA__wallet_directory>/testCAroot.cer -pwd
$ orapki wallet add -wallet <cluster_wallet_directory> -user_cert -cert
<cluster_wallet_directory>/testuser.cer
```

d. Check the finished cluster wallet.

```
$ orapki wallet display -wallet <cluster_wallet_directory>

Requested Certificates:
User Certificates:
Subject:        CN=testuser
Trusted Certificates:
Subject:        CN=test CA,O=test,C=c
```

At this point, you are ready to copy the finished cluster wallet to each node of the cluster.

3. Create the client wallet.

a. Create a client wallet with the root certificate (`testCAroot.cer`).

To make a successful TLS connection, the client only requires the CA certificate of the server's certificate.

```
$ orapki wallet create -wallet client_wallet_file_directory -auto_login
$ orapki wallet add -wallet client_wallet_file_directory -trusted_cert -
cert <CA__wallet_directory>/testCAroot.cer
```

b. Display the contents of the client wallet.

```
$ orapki wallet display -wallet client_wallet_file_directory

Requested Certificates:
User Certificates:
Trusted Certificates:
Subject:        CN=test CA,O=test,C=c
```

21.4.1.4 Step 4: Create a Wallet in Each Node of the Oracle RAC Cluster

After you have created the cluster wallet, you can copy it to each node of the Oracle Real Applications (Oracle RAC) cluster.

Ensure that each node is accessible by both the Oracle Real Application Clusters (Oracle RAC) database server (process monitor) and by the scan and local listeners that normally run from the GI home.

1. Copy the PKCS#12 wallet (`ewallet.p12`) file that you created in the previous section to each node in the cluster.
2. In each node, create an auto-login wallet (`cwallet.sso`).

   The `cwallet.sso` file is an obfuscated mirror copy of the `ewallet.p12` and is the file that the database server and its listeners accesses. If you create the `cwallet.sso` on the Oracle RAC cluster, then you can copy it along with the `ewallet.p12` file to the wallet directory on each node. You can also create the `cwallet.sso` file on each node separately if `ewallet.p12` file is already in place. Run the following command in the same location as the `ewallet.p12` file:

   ```
   $ orapki wallet create -wallet wallet_file_location -auto_login
   Enter wallet password: ewallet_password
   ```

Related Topics

- Oracle Real Application Clusters Components That Need Certificates
  Specific components in Oracle Real Application Clusters (Oracle RAC) need certificates when you configure Transport Layer Security (TLS) connections.

21.4.1.5 Step 5: Define Wallet Locations in the listener.ora and sqlnet.ora Files

To enable the database server and listeners to access the wallets, you must define the wallet locations in the `listener.ora` and `sqlnet.ora` files.

1. Modify the `listener.ora` file in the Grid home of every node.

   ```
   SSL_CLIENT_AUTHENTICATION = FALSE

   WALLET_LOCATION =
     (SOURCE =
       (METHOD = FILE)
       (METHOD_DATA =
         (DIRECTORY = wallet_file_location)
   ```

2. In the `sqlnet.ora` file in the Oracle Database home, and the Grid home, of each cluster node, add the following information:

   ```
   SQLNET.AUTHENTICATION_SERVICES = (BEQ, TCP, TCPS)

   SSL_CLIENT_AUTHENTICATION = FALSE

   WALLET_LOCATION =
     (SOURCE =
       (METHOD = FILE)
       (METHOD_DATA =
   ```

```
    (DIRECTORY = wallet_file_location)
  )
)
```

21.4.1.6 Step 6: Restart the Database Instances and Listeners

With the wallets in place and the `*.ora` files edited, you must restart the database server and listener processes so that they pick up the new settings.

The restart process will also enable the Oracle Real Application Clusters (Oracle RAC) instances where you set the `LOCAL_LISTENER` parameter earlier.

- In any cluster node, use the `srvctl` utility to restart the database server and listener processes.

  For example:

```
$ srvctl stop listener
$ srvctl start listener

$ srvctl stop scan_listener
$ srvctl start scan_listener

$ srvctl stop database -d db_name
$ srvctl start database -d db_name
```

21.4.1.7 Step 7: Test the Cluster Node Configuration

To test the cluster node configuration, you can create a connect descriptor for the node and then try to connect to this node.

1. In any cluster node, create a connect descriptor in the `tnsnames.ora` file that uses the scan listener TCPS endpoint.

   For example, for a TCPS endpoint called `dbssl`:

```
DBSSL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCPS)(HOST = scan_name)(PORT = port_2))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = service_name)
    )
  )
```

2. Use SQL*Plus to try to connect to this TCPS endpoint.

   For example:

```
sqlplus user_name/@dbssl
Enter password: password
```

21.4.1.8 Step 8: Test the Remote Client Configuration

After you have tested the wallet on the Oracle Real Applications (Oracle RAC) cluster nodes, you are ready to test the remote client configuration.

1. In every remote client `sqlnet.ora` file on the cluster node, define a wallet directory.

```
WALLET_LOCATION =
  (SOURCE =
    (METHOD = FILE)
    (METHOD_DATA =
      (DIRECTORY = wallet_file_location)
    )
  )
```

2. Move the client wallet, that you created during the setup of SSL wallets and certificates, to the client wallet directory. The wallet directory should have an `ewallet.p12` file and a `cwallet.sso` file.

   Display the contents of the wallet to ensure that the wallet directory is setup correctly.

```
$ orapki wallet display -wallet <wallet_file_location>
```

3. In the `tnsnames.ora` file, create a connect descriptor that uses the scan listener TCPS endpoint.

   For example:

```
DBSSL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCPS)(HOST = scan_name)(PORT = port_2))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = service_name)
    )
  )
```

4. Use SQL*Plus to try to connect to this TCPS endpoint. Enter the password when prompted.

   For example:

```
sqlplus user_name/@dbssl
```

21.5 Troubleshooting the Transport Layer Security Configuration

Common errors may occur while you use the Oracle Database Transport Layer Security.

A utility is available through My Oracle Support to review and provide feedback on your TLS client and server configuration. See DBSecChk Utility 2.0.0.5 (Doc ID 3066006.1).

It may be necessary to enable Oracle Net tracing to determine the cause of an error. For information about setting tracing parameters to enable Oracle Net tracing, refer to Tracing Error Information for Oracle Net Services in the *Oracle Database Net Services Administrator's Guide*.

ORA-28759: Failure to Open File
Cause: The system could not open the specified file. Typically, this error occurs because the wallet cannot be found.

Action: Check the following:

- Ensure that the correct wallet location is specified in the `sqlnet.ora` file. This should be the same directory location where you saved the wallet.
- Enable Oracle Net tracing to determine the name of the file that cannot be opened and the reason.
- Ensure that auto-login was enabled when you saved the wallet, using `orapki` or `mkstore`. The `mkstore` wallet management command line tool is deprecated with Oracle Database 23ai, and can be removed in a future release.

ORA-28786: Decryption of Encrypted Private Key Failure
Cause: An incorrect password was used to decrypt an encrypted private key. Frequently, this happens because an auto-login wallet is not being used.

Action: Use `orapki` to turn the auto-login feature on for the wallet. Then save the wallet again. For example:

```
orapki wallet create -wallet wallet_file_location -auto_login
```

If the auto-login feature is not being used, then enter the correct password.

ORA-28858: SSL Protocol Error
Cause: This is a generic error that can occur during TLS handshake negotiation between two processes.

Action: Enable Oracle Net tracing and attempt the connection again to produce trace output. Then contact Oracle customer support with the trace output.

ORA-28859 SSL Negotiation Failure
Cause: An error occurred during the negotiation between two processes as part of the TLS protocol. This error can occur when two sides of the connection do not support a common cipher suite.

Action: Check the following:

- Check the `sqlnet.ora` file to ensure that the TLS versions on both the client and the server match, or are compatible. For example, if the server accepts only TLS 1.3 and the client accepts only TLS 1.2, then the TLS connection will fail.
- Check what cipher suites are configured on the client and the server, and ensure that compatible cipher suites are set on both.

  If the error still persists, then enable tracing and attempt the connection again. Contact Oracle Support with the trace output.

> **See Also:**
>
> Specifying TLS Protocol and TLS Cipher Suites for details about setting compatible cipher suites on the client and the server

> **Note:**
>
> If you do not configure any cipher suites, then all available cipher suites are enabled.

ORA-28862: SSL Connection Failed
Cause: This error occurred because the peer closed the connection.

Action: Check the following:

- Ensure that the correct wallet location is specified in the `sqlnet.ora` file so the system can find the wallet.
- Ensure that cipher suites are set correctly in the `sqlnet.ora` file. Sometimes this error occurs because the `sqlnet.ora` has been manually edited and the cipher suite names are misspelled. Ensure that case sensitive string matching is used with cipher suite names.
- Ensure that the TLS versions on both the client and the server match or are compatible. Sometimes this error occurs because the TLS version specified on the server and client do not match. For example, if the server accepts only TLS 1.3 and the client accepts only TLS 1.2, then the TLS connection will fail.
- For more diagnostic information, enable Oracle Net tracing on the peer.

ORA-28865: SSL Connection Closed
Cause: The TLS connection closed because of an error in the underlying transport layer, or because the peer process quit unexpectedly.

Action: Check the following:

- Ensure that the TLS versions on both the client and the server match, or are compatible. Sometimes this error occurs because the TLS version specified on the server and client do not match. For example, if the server accepts only TLS 1.3 and the client accepts only TLS 1.2, then the TLS connection will fail.
- Enable Oracle Net tracing and check the trace output for network errors.

ORA-28868: Peer Certificate Chain Check Failed
Cause: When the peer presented the certificate chain, it was checked and that check failed. This failure can be caused by a number of problems, including:

- One of the certificates in the chain has expired.
- A certificate authority for one of the certificates in the chain is not recognized as a trust point.
- The signature in one of the certificates cannot be verified.

Action: Open your wallet and check the following:

- Ensure that all of the certificates installed in your wallet are current (not expired).
- Ensure that a certificate authority's certificate from your peer's certificate chain is added as a trusted certificate in your wallet.

ORA-28885: No certificate with the required key usage found.
Cause: Your certificate was not created with the appropriate X.509 version 3 key usage extension.

Action: Create the certificate with the appropriate X.509 version 3 key usage extension. For example:

```
orapki wallet add -wallet user_wallet -asym_alg ECC -eccurve p384 -sign_alg
ecdsasha384 -dn 'cn=user_ecc,c=us' -pwd welcome1 -addext_ku digitalSignature
```

You may add more key usages than just `digitalSignature`, for example:

```
-addext_ku
digitalSignature,nonRepudiation,keyEncipherment,dataEncipherment,keyAgreement,
keyCertSign,cRLSign,encipherOnly,decipherOnly
```

ORA-29019: The Protocol Version is incorrect
Cause: There is a protocol version mismatch between the two peers.

Action: Specify the correct protocol version or unset `SSL_VERSION` in the product's configuration file.
The error code is shown in the trace: `[DATE_AND_TIME] ntzdosecneg: SSL handshake failed with error 29019`.

ORA-29024: Certificate Validation Failure
Cause: The certificate sent by the other side could not be validated. This may occur if the certificate has expired, has been revoked, or is invalid for any other reason.

Action: Check the following:

- Check the certificate to determine whether it is valid. If necessary, get a new certificate, inform the sender that their certificate has failed, or resend.
- Check to ensure that the server's wallet has the appropriate trust points to validate the client's certificate. If it does not, then use `orapki` to import the appropriate trust point into the wallet.
- Ensure that the certificate has not been revoked and that certificate revocation list (CRL) checking is turned on. For details, refer to Configuring Certificate Validation with Certificate Revocation Lists

ORA-29223: Cannot Create Certificate Chain
Cause: A certificate chain cannot be created with the existing trust points for the certificate being installed. Typically, this error is returned when the peer does not give the complete chain and you do not have the appropriate trust points to complete it.

Action: Use `orapki` to install the trust points that are required to complete the chain.

21.6 Migrating to and Configuring Transport Layer Security Version 1.3

Version 1.3 of Transport Layer Security (TLS) provides stronger security and faster TLS handshakes, when compared to previous versions of TLS.

TLS version 1.3 is supported and enabled by default with 23ai when both the database server and client are version 23ai.

If your environment does not specify the `SSL_VERSION` parameter in the configuration files, then TLS version 1.3 is enabled by default. If the `SSL_CIPHER_SUITES` parameter is not explicitly

configured, TLS 1.3 cipher suites get automatically picked. The product is designed to pick the strongest TLS version and the strongest available cipher in that version.

The enhancements in Transport Layer Security (TLS) version 1.3 may affect current TLS configurations if one or both of the following parameters are specified.

- `SSL_VERSION`: Remove this parameter from the configuration files to enable all supported TLS versions, or include the string "TLSv1.3" in the value specified
  For example,

  ```
  SSL_VERSION = (TLSv1.3, TLSv1.2)
  ```

- `SSL_CIPHER_SUITES`: Remove this parameter from the configuration files to enable all supported TLS cipher suites, or include one or more of the TLS version 1.3 cipher suites
  For example,

  ```
  SSL_CIPHER_SUITES = (TLS_AES_256_GCM_SHA384, TLS_AES_128_GCM_SHA256,
  TLS_AES_128_CCM_SHA256)
  ```

Related Topics

- Configuring TLS Protocol Versions
  The `SSL_VERSION` and `TLS_DISABLE_VERSION` parameters define the protocol version of TLS that is enforced at the end point of the component where they are specified.
- Troubleshooting Transport Layer Security Errors
  Oracle provides a utility to help troubleshoot PKI certificate configurations as well as additional guidance below. A utility is available through the support website to review and provide feedback on your PKI certificate authentication client and server configuration.
- Allowing Certificates from Earlier Algorithms
  You can use certificates that were associated with earlier deprecated (and weaker) algorithms by setting the `ALLOWED_WEAK_CERT_ALGORITHMS` `sqlnet.ora` or `listener.ora` parameter.