Redbooks

ibm.com/redbooks

IBM DB2 for z/OS: Configuring TLS/SSL for Secure Client/Server Communications

Chris Meyer

Derek Tempongko

Information Management

Redpaper

IBM Db2 for z/OS: Configuring TLS/SSL for Secure Client/Server Communications

This IBM® Redpaper publication provides information about how to set up and configure IBM Db2® for z/OS® with Transport Layer Security (TLS), which is the modern version of Secure Sockets Layer (SSL). This configuration is accomplished by using the IBM z/OS Communications Server Application Transparent Transport Layer Security (AT-TLS) services.

This paper also describes the steps for configuring TLS/SSL support for the IBM Data Server Driver Package (DS Driver) for IBM Data Server Provider for .NET, Open Database Connectivity (ODBC), and Call Level Interface clients to access a Db2 for z/OS server. In addition, this paper provides information about configuring that same support for the Java Database Connectivity (JDBC) and Structured Query Language for Java (SQLJ for Type 4 connectivity) clients.

The information that is provided is applicable to Db2 12 for z/OS and Db2 11 for z/OS.

Although we use z/OS V2R4 as the referenced release in this paper, the instructions, except for a TLSv1.3 configuration, are valid for releases as early as z/OS V2R1.

Throughout the paper, we reference z/OS Security Server or IBM Resource Access Control Facility (IBM RACF®) in various contexts. It should be understood that anywhere we mention RACF, it implies any System Authorization Facility (SAF)-compliant external security manager.

The intended audience for this paper includes network administrators, security administrators, and database administrators who want to set up and configure TLS/SSL support for Db2 for z/OS.

This paper provides information about the following topics:

- Overview of AT-TLS
- Configuring Db2 for z/OS as a server with TLS/SSL support
- Configuring Db2 for z/OS as a requester with TLS/SSL support
- Configuring Java applications by using IBM DS Driver for JDBC and SQLJ to use TLS/SSL
- Configuring the IBM DS Driver non-Java interfaces: Command-line interface, ODBC, and .NET

- Configuring remote client applications to use TLS/SSL through a Db2 Connect server for Linux, UNIX, and Windows
- Client access to Db2 by using TLS/SSL client authentication
- Using Windows truststore and Windows keystore

This paper presents more information about the more general contents of *Security Functions of IBM DB2 10 for z/OS*, SG24-7959.

Overview of AT-TLS

TLS is a client/server cryptographic protocol that is based on the SSL specifications that were developed by Netscape Corporation in the 1990s for securing communications that use Transmission Control Protocol/Internet Protocol (TCP/IP) sockets. Unless stated otherwise, the terms TLS and SSL are used interchangeably in this document.

It is important to understand the basics of the TLS handshake and how digital certificates are used in TLS/SSL before reading the rest of this paper. If you are not familiar with these topics, see one of the many TLS/SSL primers that are available on the internet.

The TLS protocol runs at the application level (like its predecessor SSL). Therefore, an application typically must be designed and coded to use TLS/SSL protection. On z/OS, the System SSL component of the Cryptographic Services element implements the full suite of TLS/SSL protocols (up to and including TLSv1.3 and TLSv1.2 at of the time of writing) through a robust set of application programming interfaces (APIs) for z/OS C and C++ applications.

To make TLS/SSL more accessible to z/OS applications regardless of their source programming language, z/OS Communications Server provides Application Transparent TLS (AT-TLS). AT-TLS invokes TLS/SSL primitives in the TCP layer of the TCP/IP stack on behalf of application programs based on policy rules that describe the application traffic and how to protect it. With AT-TLS, applications that are written in almost any language that is supported on z/OS can enjoy full TLS/SSL protection without requiring source code changes.

Db2 relies on AT-TLS to secure its TCP connections from and to remote systems.

AT-TLS policy is read, parsed, and installed into the TCP/IP stack by the z/OS Communications Server Policy Agent (PAGENT), which implements policy-based networking for the z/OS environment. For more information about policy-based networking, see *z/OS V2R4 Communications Server: IP Configuration Guide*, SC27-3650.

Although this paper describes AT-TLS policies by using the underlying PAGENT syntax, the preferred method for creating and maintaining AT-TLS policies is through the z/OSMF -based IBM Network Configuration Assistant (NCA) web UI. For more information about the NCA, see the IBM Network Configuration Assistant for z/OS Communications Server AT-TLS tutorial, found at:

`https://www.ibm.com/docs/en/zos/2.4.0?topic=folder-getting-started-tutorial-tls`

Because AT-TLS is largely transparent to Db2, Db2 continues to send and receive clear text data over its sockets while an active AT-TLS policy directs TCP/IP to invoke the required System SSL services to protect the data that is being transmitted over the network.

Figure 1 shows the flow of Db2 for z/OS acting as a server by using AT-TLS to protect the data exchanges over the network with a remote client system that also supports TLS.

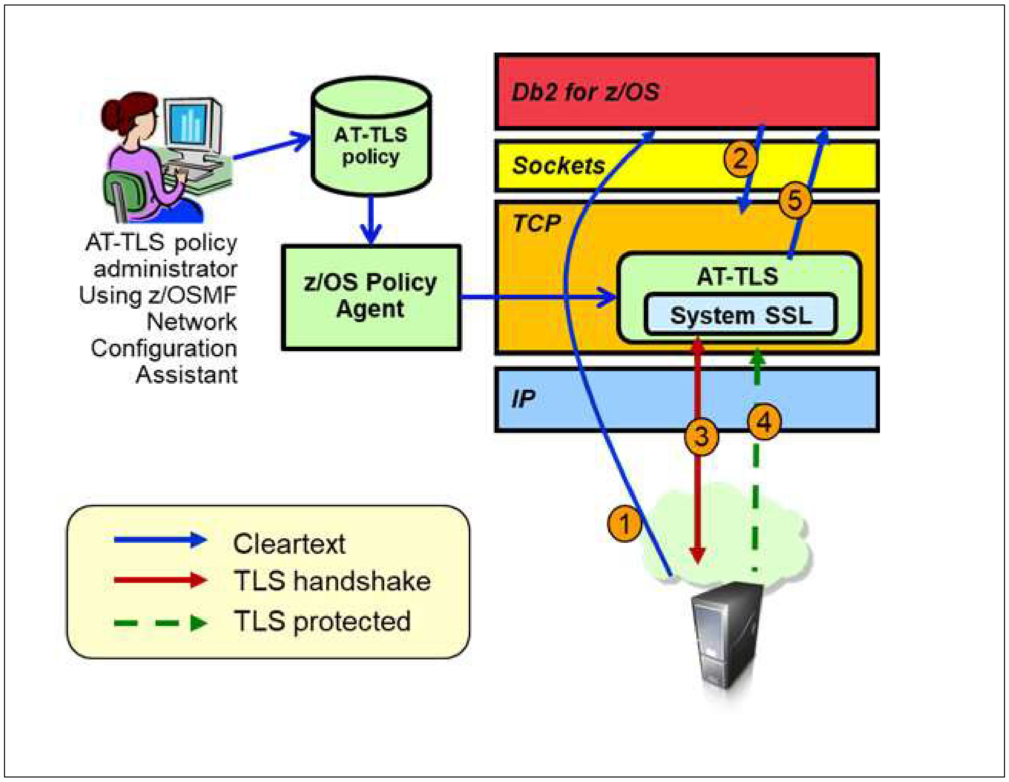


Figure 1   Flow of Db2 for z/OS as a server by using AT-TLS

The diagram in Figure 1 illustrates the following flow:

1. After the PAGENT successfully reads the AT-TLS policy and installs it into the TCP/IP stack, the client's TCP connection to the Db2 server is established by using a standard TCP 3-way handshake.
2. After accepting the new connection, Db2 issues a read request on the socket. The TCP layer checks the AT-TLS policy and sees that AT-TLS protection is configured for this connection. Therefore, TCP prepares for the client-initiated TLS handshake.
3. The client initiates the TLS handshake, and the TCP layer invokes System SSL to perform its portion of the TLS handshake under the identity of the Db2 server. System SSL and the TLS software at the client exchange handshake messages according to the relevant Requests for Comments (RFCs) to complete the TLS session.
4. The Db2 client sends data to the Db2 server under protection of the new TLS session.
5. The TCP layer receives the inbound data, where AT-TLS invokes System SSL to decrypt the data, and then it delivers the clear text inbound data to the Db2 server.

Likewise, when the Db2 server sends data to the Db2 client, AT-TLS takes the clear text outbound data, calls System SSL to encrypt it, and then the encrypted data is sent to the client.

Configuring Db2 for z/OS as a server with TLS/SSL support

There are two approaches to configuring AT-TLS and PAGENT on z/OS:

- The preferred approach is to use the z/OS Management Facility (z/OSMF) NCA for z/OS Communications Server. The NCA is a GUI-based tool that greatly simplifies the setup, configuration, and deployment of z/OS Communications Server policy-based technologies, including AT-TLS.

  For more information, see the tutorial and help windows in the Configuration Assistant tool. For a complete example of configuring AT-TLS by using the Configuration Assistant, see Chapter 12, "Application Transparent Transport Layer Security" in *IBM z/OS V2R2 Communications Server TCP/IP Implementation: Volume 4 Security and Policy-Based Networking*, SG24-8363.
- The alternative approach is to hand-code all of the necessary job control language (JCL), RACF directives, configuration files, and policy files. To provide you with greater insight into the PAGENT and AT-TLS configuration, this approach is the one that we describe in detail for the remainder of this document.

Security of REST service connections

Db2 supports HTTPS service requests by configuring Db2 for z/OS as a server with TLS/SSL support. To accomplish this task, complete the following steps:

1. Configuring the PAGENT as a started task in z/OS.
2. Defining security authorizations for the PAGENT.
3. Defining the PAGENT configuration files.
4. Configuring AT-TLS.
5. Defining the AT-TLS policy rules.
6. Creating and configuring digital certificates.
7. Configuring a secure port for the Db2 for z/OS server.

We describe these steps in the following sections.

Configuring the PAGENT as a started task in z/OS

The PAGENT runs as a UNIX process, so it can be started either from the UNIX System Services shell or as a z/OS started task. In our example, we use the z/OS started task procedure to start the PAGENT.

To start the PAGENT as a z/OS started task, you can use the JCL procedure that is shown in Example 1.

Example 1 PAGENT JCL procedure

```
//PAGENT   PROC
//PAGENT   EXEC PGM=PAGENT,REGION=OK,TIME=NOLIMIT,
//       PARM='POSIX(ON) ALL31(ON) ENVAR("_CEE_ENVFILE=DD:STDENV")/'
//*
//* For information on the above environment variables, refer to the
//* IP CONFIGURATION GUIDE. Other environment variables can also be
//* specified through STDENV.
//*
//STDENV   DD DSN=SYS1.TCPPARMS(PAENV),DISP=SHR
//*
//* Output that is written to stdout and stderr goes to the data set or
//* file specified with SYSPRINT or SYSOUT, respectively. But
//* normally, PAGENT doesn't write output to stdout or stderr.
//* Instead, output is written to the log file, which is specified
//* by the PAGENT_LOG_FILE environment variable, and defaults to
//* /tmp/pagent.log. However, when the -d parameter is specified
//* output is also written to stdout.
//*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
```

You can also find a sample started task procedure for PAGENT in `TCPIP.SEZAINST(EZAPAGSP)`.

You can use environment variables that are either configured in an IBM Multiple Virtual Storage (IBM MVS) sequential or partitioned data set, or a z/OS UNIX file that is specified by the `STDENV` data definition (DD) to run with the required configuration. In our example, we configured the environment variables for PAGENT in partitioned data set `SYS1.TCPPARMS`, member `PAENV`, as shown in Example 2.

Example 2 SYS1.TCPPARMS(PAENV) data set containing PAGENT environment variables

```
TZ=PST8PDT7
PAGENT_CONFIG_FILE=//'SYS1.TCPPARMS(PAGENT)'
PAGENT_LOG_FILE=SYSLOGD
```

The following environment variables are used in Example 2:

TZ	Specifies the local time zone for the PAGENT process. Here, it is set to `PST8` (Pacific Standard Time GMT-08:00) and `PDT7` (Pacific Daylight Saving Time GMT-07:00).
PAGENT_CONFIG_FILE	Specifies the PAGENT configuration file to use. The configuration file is the PAGENT member of the `SYS1.TCPPARMS` data set in this example.
PAGENT_LOG_FILE	Specifies the logging destination that is used by PAGENT. It is set to log PAGENT messages to `SYSLOGD`.

Before you can start PAGENT, you must define the appropriate security authorizations, as described in the next section.

Defining security authorizations for the PAGENT

The policies that are managed by the PAGENT can affect system operation significantly, s you must restrict the list of z/OS user IDs (UIDs) under which PAGENT is allowed to run. To do this task, you must define certain resources and controls in the system's security manager product, such as RACF.

To set up the PAGENT's security definitions in RACF, complete the following steps:

1. Defining the PAGENT started task in RACF.
2. Defining the PAGENT UID.
3. Associating the PAGENT UID with the PAGENT started task.
4. Giving authorized users access to manage the PAGENT started task.
5. Restrict access to the **`pasearch`** UNIX command.

Defining the PAGENT started task in RACF

In this example, the PAGENT runs under z/OS as a started task that is named PAGENT. To set up the PAGENT started task to RACF, you must define a profile for it in the RACF generic resource class that is called `STARTED` by using the **`RDEFINE`** command.

> **SETROPTS:** In Example 3, we include the **`SETROPTS`** commands for completeness. Running these commands when the `STARTED` class is already activated has no effect.

Example 3 shows the RACF commands that are used to set up the PAGENT started task.

Example 3 RACF commands to define the PAGENT started task in RACF

```
//DAEMONS  EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 SETROPTS CLASSACT(STARTED)
 SETROPTS RACLIST(STARTED)
 SETROPTS GENERIC(STARTED)
 RDEFINE  STARTED  PAGENT.*
 SETROPTS RACLIST(STARTED) REFRESH
 SETROPTS GENERIC(STARTED) REFRESH
/*
```

If you also want to log messages through `SYSLOGD`, include an **`RDEFINE`** command to define a profile for `SYSLOGD` to the `STARTED` class, as shown in the following example:

```
RDEFINE  STARTED  SYSLOGD.*
```

Defining the PAGENT UID

In this example, the PAGENT runs under the z/OS UID PAGENT. PAGENT often runs under a UID with z/OS UNIX superuser authority (the z/OS UNIX UID for this user must be set to 0), although this authority is not required. To run under a UID that does not have superuser authority, see “Other considerations when starting the Policy Agent” in *z/OS V2R4 Communications Server: IP Configuration Guide*, SC27-3650. Additionally, you must assign a default group (`DFLTGRP`) for the UID.

Example 4 shows the RACF command that is used to define a UID that is called PAGENT to a default group that is called `OMVSGRP` with an OMVS segment with a UNIX UID of 0.

Example 4 RACF command to define a UID for the PAGENT started task

```
//PAUSER   EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  ADDUSER  PAGENT   DFLTGRP(OMVSGRP)  OMVS(UID(0)  HOME('/'))
/*
```

If your security administrator defined the `SHARED.IDS` profile in the `UNIXPRV` class, the UID value must be unique. If you want to use a UID value that is already in use, add the **`SHARED`** keyword to the UID parameter of the **`ADDUSER`** command to indicate that you intend to share the UID value across z/OS UIDs.

If you are also going to log messages to `SYSLOGD`, define a UID with superuser authority for the SYSLOGD started task, as shown in the following example:

```
ADDUSER  SYSLOGD   DFLTGRP(OMVSGRP)  OMVS(UID(0)  HOME('/'))
```

Associating the PAGENT UID with the PAGENT started task

Use the RACF **RALTER** command to associate the PAGENT UID that was created to the PAGENT started task, as illustrated in Example 5.

Example 5 RACF command to associate the UID with the PAGENT started task

```
//ASCPAUSR EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  RALTER STARTED PAGENT.* STDATA(USER(PAGENT))
/*
```

If you are also going to log messages to `SYSLOGD`, you must associate the `SYSLOGD` UID with the `SYSLOGD` started task, as shown in the following example:

```
RALTER STARTED PAGENT.* STDATA(USER(SYSLOGD))
```

Giving authorized users access to manage the PAGENT started task

To restrict management access to the PAGENT started task, you must define a profile that is named `MVS.SERVMGR.PAGENT` in the RACF resource class `OPERCMDS`, and give only authorized users access to this profile. Example 6 shows the RACF commands that are used to achieve this access.

Example 6 RACF commands to give authorized users access to manage the PAGENT started task

```
//PERMITPA EXEC PGM=IKJEFT01
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 SETROPTS CLASSACT(OPERCMDS)
 SETROPTS RACLIST (OPERCMDS)
 RDEFINE  OPERCMDS (MVS.SERVMGR.PAGENT) UACC(NONE)
 PERMIT   MVS.SERVMGR.PAGENT CLASS(OPERCMDS) ACCESS(CONTROL) -
          ID(PAGENT)
 SETROPTS RACLIST(OPERCMDS) REFRESH
/*
```

Restricting access to the pasearch UNIX command

Use the **pasearch** UNIX command to display policy definitions. The output from this command indicates whether policy rules are active, and it shows the parsed results of the policy definition attributes.

The **pasearch** output can be used to verify that the policies are defined correctly. However, only users with a specific need to know this information should be able to see the policy definitions. To restrict access to the **pasearch** command, a profile is defined in the RACF `SERVAUTH` resource class. This type of profile can be defined for each TCP/IP stack (**TcpImage**) and policy type (ptype):

```
EZB.PAGENT.sysname.TcpImage.ptype
```

In this example, these variables have the following definitions:

sysname	The z/OS system name.
TcpImage	The name of the TCP/IP stack (its procedure name) to which policy information is to be restricted.
ptype	The type of policy that is being requested. The following types are possible:

QoS	Quality of service (QoS) policy
IDS	Intrusion Detection Services (IDS) policy
IPsec	IP packet filtering and IPsec protocol policy
TTLS	AT-TLS policy
Routing	Policy-Based Routing policy
CFGSERV	TCP/IP profile information

Tip: Wildcard use is supported in segments of RACF profiles. For example, the `EZB.PAGENT.UTEC224.*.*` profile controls **pasearch** access to all policy types for all TCP/IP stacks on the z/OS system that is named UTEC224.

Example 7 shows the RACF commands that are used to restrict access to the **pasearch** UNIX command. In this example, only z/OS UIDs `USRT001` and `USRT002` are permitted to use **pasearch** for all types of policies for the TCP/IP stack that is named `TCPIP` on the z/OS system that is named `UTEC224`.

Example 7 RACF commands to restrict access to the pasearch UNIX command

```
//PAGNACC  EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 RDEFINE  SERVAUTH EZB.PAGENT.UTEC224.TCPIP.* UACC(NONE)
 PERMIT   EZB.PAGENT.UTEC224.TCPIP.* CLASS(SERVAUTH) -
          ID(USRT001) ACCESS(READ)
 PERMIT   EZB.PAGENT.UTEC224.TCPIP.* CLASS(SERVAUTH) -
          ID(USRT002) ACCESS(READ)
 SETROPTS GENERIC(SERVAUTH) REFRESH
 SETROPTS RACLIST(SERVAUTH) REFRESH
/*
```

Now that you set up all the security authorizations for the PAGENT, you must configure the policy for AT-TLS.

Defining the PAGENT configuration files

The PAGENT is responsible for reading policies from MVS data sets or from z/OS UNIX file system files. Before we can define the AT-TLS policies, we must configure certain operational characteristics of the PAGENT in the main configuration file. The main configuration file can contain the following statements:

- **`TcpImage`** statement
- **`LogLevel`** statement

The following two sections describe the configuration steps:

- Defining the TcpImage statements
- Defining the appropriate logging level

Defining the TcpImage statements

The **`TcpImage`** statement specifies a TCP/IP stack and its associated stack-specific configuration file. There can be one **`TcpImage`** statement for each TCP/IP stack on the z/OS system. If a configuration file name is specified, it identifies the file that contains image-specific configuration definitions for AT-TLS policies. If the file name is not specified, the main configuration file contains the image-specific configuration for that stack.

Depending on your environment, you can define the **`TcpImage`** statement in the following ways:

- If your environment is configured with multiple TCP/IP stacks, specify **`TcpImage`** statements identifying an image-specific configuration file for each TCP/IP stack. Figure 2 shows this type of configuration.

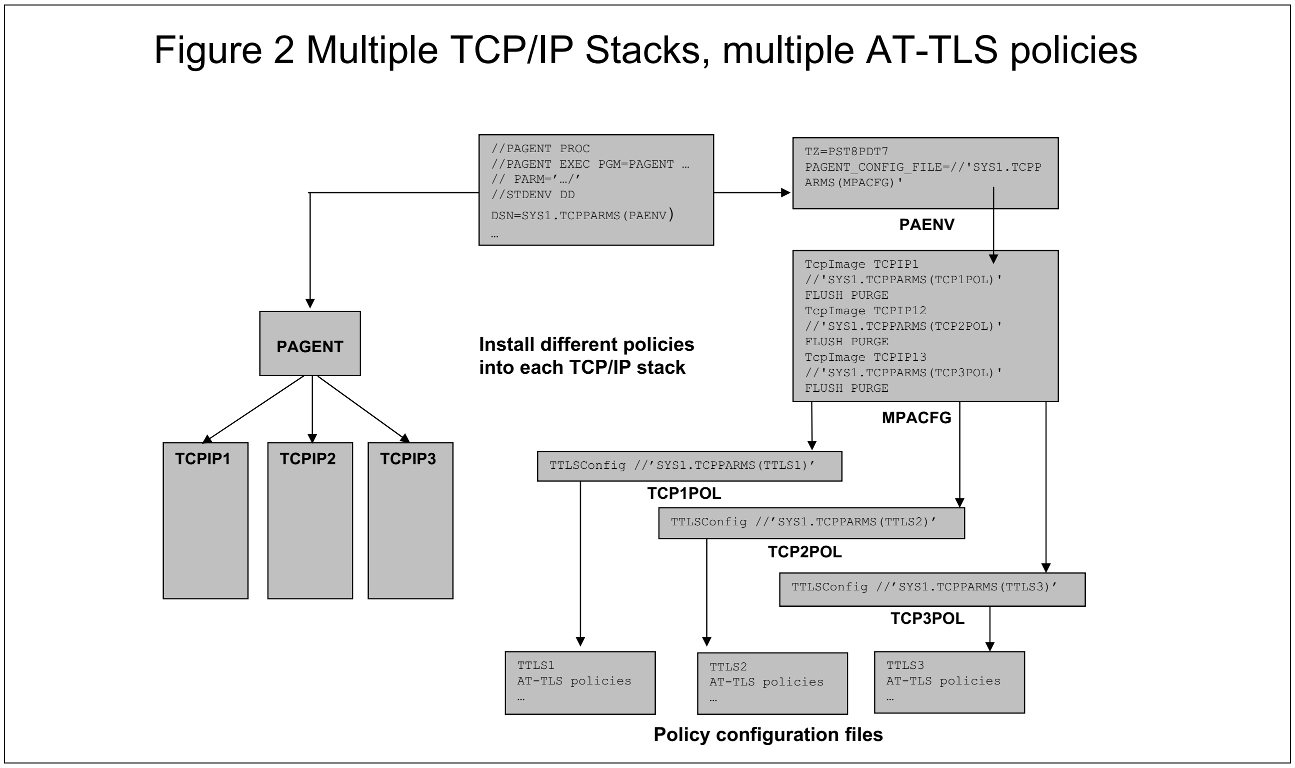


Figure 2 Multiple TCP/IP Stacks, multiple AT-TLS policies
PAGENT
TCPIP1
TCPIP2
TCPIP3
Install different policies into each TCP/IP stack
PAENV
MPACFG
TCP1POL
TCP2POL
TCP3POL
Policy configuration files

```
//PAGENT PROC
//PAGENT EXEC PGM=PAGENT …
// PARM='…/'
//STDENV DD
DSN=SYS1.TCPPARMS(PAENV)
…
```

```
TZ=PST8PDT7
PAGENT_CONFIG_FILE=//'SYS1.TCPP
ARMS(MPACFG)'
```

```
TcpImage TCPIP1
//'SYS1.TCPPARMS(TCP1POL)'
FLUSH PURGE
TcpImage TCPIP12
//'SYS1.TCPPARMS(TCP2POL)'
FLUSH PURGE
TcpImage TCPIP13
//'SYS1.TCPPARMS(TCP3POL)'
FLUSH PURGE
```

```
TTLSConfig //'SYS1.TCPPARMS(TTLS1)'
```

```
TTLSConfig //'SYS1.TCPPARMS(TTLS2)'
```

```
TTLSConfig //'SYS1.TCPPARMS(TTLS3)'
```

```
TTLS1
AT-TLS policies
…
```

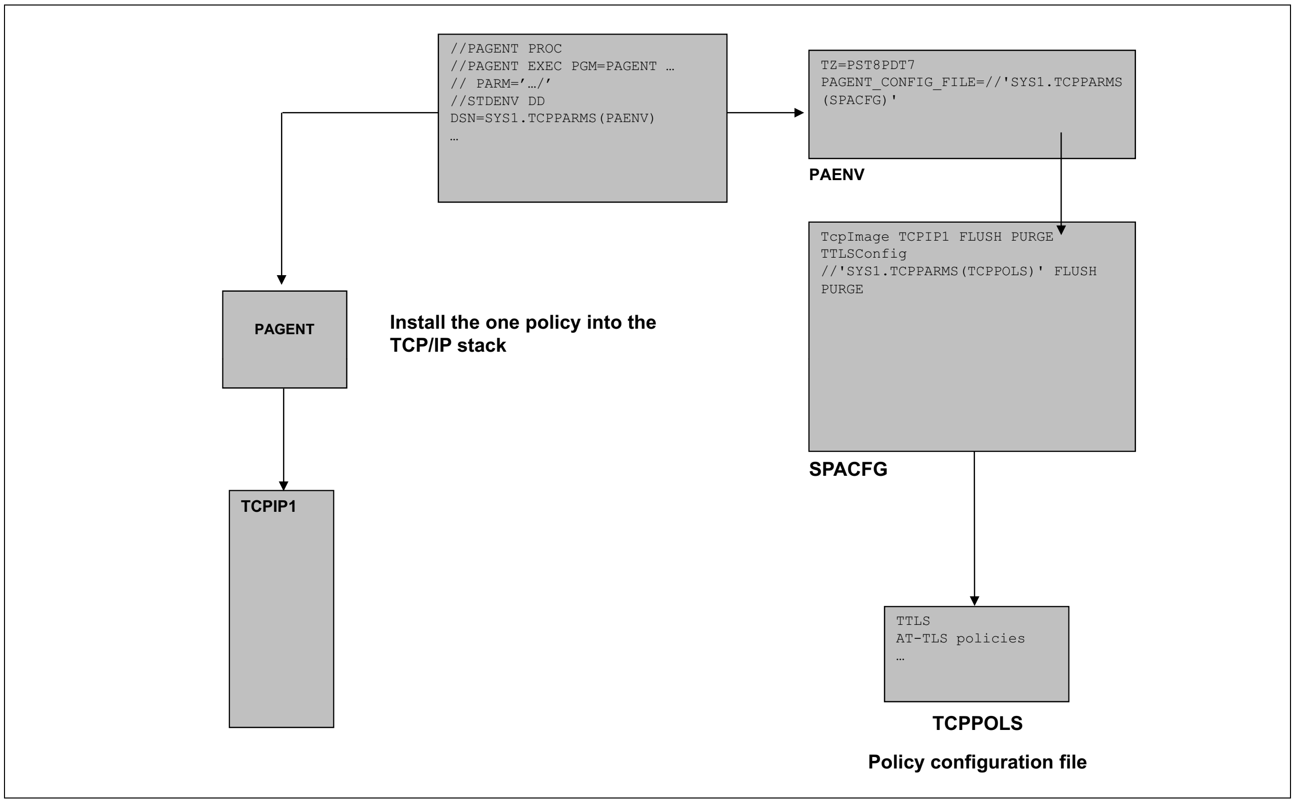
```
TTLS2
AT-TLS policies
…
```

```
TTLS3
AT-TLS policies
…
```

Figure 2 Multiple TCP/IP stacks and multiple AT-TLS policies

- If your environment is configured with a single TCP/IP stack, only one AT-TLS policy definition is needed. In this case, you specify the **`TcpImage`** statement for the single TCP/IP stack, and omit the specification of the image-specific configuration file. The main configuration file contains a **`TTLSConfig`** statement that identifies a single policy file for the TCP/IP stack.

Figure 3 shows this type of configuration.



PAGENT
Install the one policy into the TCP/IP stack
TCPIP1
PAENV
SPACFG
TCPPOLS
Policy configuration file

```
//PAGENT PROC
//PAGENT EXEC PGM=PAGENT …
// PARM='…/'
//STDENV DD
DSN=SYS1.TCPPARMS(PAENV)
…
```

```
TZ=PST8PDT7
PAGENT_CONFIG_FILE=//'SYS1.TCPPARMS
(SPACFG)'
```

```
TcpImage TCPIP1 FLUSH PURGE
TTLSConfig
//'SYS1.TCPPARMS(TCPPOLS)' FLUSH
PURGE
```

```
TTLS
AT-TLS policies
…
```

Figure 3 Single TCP/IP stack and a single AT-TLS policy

It is possible that TCP/IP stacks that are configured to the PAGENT are not started or even defined. When this situation happens, the PAGENT logs an error when trying to connect to those stacks for diagnostic purposes.

A **TcpImage** statement optionally can specify the parameters **FLUSH/NOFLUSH** or **PURGE/NOPURGE**. These optional parameters determine whether policies are deleted from the associated TCP/IP stack under the conditions that are detailed in Table 1.

Table 1 How policy agent FLUSH and PURGE parameters are used

Parameter	When used	Results
FLUSH and **NOFLUSH**	Used after policies are read without errors when they are triggered by the following events: ► PAGENT start. ► **TcpImage** statement added. ► **MODIFY REFRESH** command issued.	For **FLUSH**: ► All policies are deleted from PAGENT and the TCP/IP stack before installing new policies. ► QoS policy statistics are reset to 0. For **NOFLUSH**: ► No policies are deleted from PAGENT or the TCP/IP stack before installing new policies. ► Policies that are removed from the configuration file are not deleted from the PAGENT or the TCP/IP stack.
PURGE and **NOPURGE**	► PAGENT shutdown. ► **TcpImage** statement deleted.	For **PURGE**: All policies are deleted from the PAGENT and the TCP/IP stack. For **NOPURGE**: ► All policies are deleted from PAGENT. ► No policies are deleted from the TCP/IP stack.

When any (or all) TCP/IP stacks are shut down, the PAGENT continues to run. When the TCP/IP stacks are restarted, active policies are reinstalled automatically.

The following example defines the main configuration file that installs an AT-TLS policy file `SYS1.TCPPARMS(TTLSPOL)` to the TCP/IP stack that is named `TCPIP` after flushing the existing policy data when the TCP/IP stack is restarted and ensuring that the policy is deleted from the stack when PAGENT shuts down:

```
TcpImage TCPIP //'SYS1.TCPPARMS(TTLSPOL)' FLUSH PURGE
```

Defining the appropriate logging level

Use the **LogLevel** statement to control the amount of information that is logged by the PAGENT. The default is to log only event, error, console, and warning messages. This level might be an appropriate level of information for a stable policy configuration, but more logging might be required to understand policy processing or debug problems when first setting up policies or when making significant changes. Specify the **LogLevel** statement with the appropriate logging level in the main configuration file.

To define the appropriate logging level in the main configuration file, specify the **LogLevel** statement keyword followed by an integer that represents the level of logging to be performed by the PAGENT. The following levels are supported:

1	SYSERR: System error messages
2	OBJERR: Object error messages
4	PROTERR: Protocol error messages
8	WARNING: Warning messages
16	EVENT: Event messages
32	ACTION: Action messages

64	INFO: Informational messages
128	ACNTING: Accounting messages
256	TRACE: Trace messages

To include more than one level of logging, specify the sum of all the selected log levels in the **LogLevel** statement. For example, to request SYSERR messages (level 1) and EVENT messages (level 16), you specify **LogLevel 17**.

LogLevel: LogLevel 63 is sufficient for debugging the most common policy and certificate errors that you might encounter during your initial AT-TLS setup. If this level is not sufficient, we advise you to set the **LogLevel** to 511. However, this more detailed level of logging can produce a significant amount of output. Consider the log size when the syslog daemon is used as the log destination.

Example 8 shows the definitions of the main configuration file for a single TCP/IP stack environment. The AT-TLS policies are defined in a separate image-specific configuration file, TTLSPOL, for the TCP/IP stack that is named TCPIP.

Example 8 PAGENT configuration file

```
# LogLevel statement
#   SYSERR, OBJERR, PROTERR, and WARNING messages are logged.
LogLevel 15
#
# TcpImage statement
#   TCP/IP image: TCPIP
#   Path to image-specific configuration file: SYS1.TCPPARMS(TTLSPOL)
#   FLUSH parameter specified to delete existing policy data in the
#   stack on PAGENT start-up or when the configuration files change.
#   PURGE parameter specified to delete active policy data from the
#   stack when PAGENT is shut down normally.
TcpImage TCPIP //'SYS1.TCPPARMS(TTLSPOL)' FLUSH PURGE
```

Next, the steps to configure AT-TLS are described.

Configuring AT-TLS

AT-TLS support is enabled by the specifying **TTLS** parameter on the **TCPCONFIG** statement in the TCP/IP profile data set. The information that is required to negotiate secure connections is provided to the TCP/IP stack by AT-TLS policies that are read, parsed, and installed by the PAGENT.

When AT-TLS is enabled and a new TCP connection is established, the AT-TLS component in the TCP layer of the stack searches for an AT-TLS rule in the policy that matches the characteristics (local and remote IP addresses, local and remote ports, connection direction, and so on) of the connection. If such a rule is found, TLS protection is applied to it according to the details that are specified in the AT-TLS action that is associated with the rule. If no such rule is found, the connection is not protected by TLS.

Tip: To *enable* TTLS without modifying `PROFILE.TCPIP` and without stopping and starting the TCP/IP stack, define a separate file that contains the **`TCPCONFIG TTLS`** statement, and issue the **`VARY TCPIP OBEYFILE`** command, as shown in this example:

1. **`OBEYFILE`** calls **`TTLSON`** in the `SYS1.TCPPARMS` partitioned data set:

   ```
   TCPCONFIG TTLS
   ```

2. On the MVS console, issue this command:

   ```
   VARY TCPIP,,O,DSN=SYS1.TCPPARMS(TTLSON)
   ```

To *disable* TTLS, perform the following actions:

1. **`OBEYFILE`** calls **`TTLSOFF`** in the `SYS1.TCPPARMS` partitioned data set:

   ```
   TCPCONFIG NOTTLS
   ```

2. On the MVS console, issue this command:

   ```
   VARY TCPIP,,O,DSN=SYS1.TCPPARMS(TTLSOFF)
   ```

Setting up TTLS stack initialization access control for AT-TLS

This step is optional, but might be required depending on your local security requirements and the services that run on your z/OS system. When AT-TLS is started during TCP/IP stack initialization, there might be a delay between the time that the stack starts and when the PAGENT successfully installs policy information into the stack. This situation can leave a window of time where connections that are intended to be protected by AT-TLS can be established without that protection.

To prevent such connections from being established while this window is open, define a profile for the `EZB.INITSTACK` resource in the RACF `SERVAUTH` resource class. The profile name can be defined by the TCP/IP stack (**`TcpImage`**):

```
EZB.INITSTACK.sysname.TcpImage
```

The variables have these definitions:

`sysname`	The system name that is assigned to the z/OS system.
`TcpImage`	The TCP/IP procedure name for the TCP/IP stack to which access is to be controlled.

With such a profile in place, you can control which applications are allowed to establish TCP connections before the AT-TLS policy is installed by the PAGENT. To do so, provide READ access to the `EZB.INITSTACK.`*`sysname`*, which is the *`TcpImage`* profile in the `SERVAUTH` class for the z/OS UIDs under which such applications run. All other applications are forced to wait until PAGENT has initialized and its policies are installed into the stack before connections can be established.

Example 9 shows the RACF commands that are used to enable the z/OS UIDs `OMVSKERN`, `PAGENT`, and `SYSLOGD` to establish TCP/IP connections before PAGENT has loaded the AT-TLS policy.

Example 9 RACF commands to set up TCP/IP stack initialization access control for AT-TLS

```
//STACKACC EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 SETROPTS CLASSACT(SERVAUTH)
 SETROPTS RACLIST (SERVAUTH)
 SETROPTS GENERIC (SERVAUTH)
```

```
 RDEFINE  SERVAUTH EZB.INITSTACK.UTEC224.TCPIP UACC(NONE)
 PERMIT   EZB.INITSTACK.UTEC224.TCPIP CLASS(SERVAUTH) -
          ID(OMVSKERN) ACCESS(READ)
 PERMIT   EZB.INITSTACK.UTEC224.TCPIP CLASS(SERVAUTH) -
          ID(PAGENT) ACCESS(READ)
 PERMIT   EZB.INITSTACK.UTEC224.TCPIP CLASS(SERVAUTH) -
          ID(SYSLOGD) ACCESS(READ)
 SETROPTS GENERIC(SERVAUTH) REFRESH
 SETROPTS RACLIST(SERVAUTH) REFRESH
/*
```

This step is optional for Db2 because Db2 does not process any TCP/IP connections until the TCP/IP stack is initialized. By then, all applicable AT-TLS policies are loaded into the TCP/IP stack.

For more information about using `EZB.INITSTACK` profiles, see "TCP/IP stack initialization access control" in the *z/OS V2R4 Communications Server: IP Configuration Guide*, SC27-3650, which includes advice about the minimum set of applications that typically are given access.

AT-TLS configuration is complete. We can now start and stop the PAGENT started task and issue the commands to refresh policies in the PAGENT.

Starting and stopping PAGENT from the z/OS console

Use the MVS **`START`** command to start the PAGENT as a started task, for example:

```
S PAGENT
```

To shut down PAGENT (normally), use the MVS **`STOP`** command, as shown in this example:

```
P PAGENT
```

If the **`PURGE`** option is specified in the policy configuration files, PAGENT deletes all policies from the TCP/IP stack when PAGENT shuts down.

Defining the AT-TLS policy rules

An AT-TLS policy configuration file contains AT-TLS rules that identify specific types of TCP connections, along with the type of TLS/SSL protection to be applied to those connections. If a matching rule is found, AT-TLS invokes System SSL to protect the connection that uses the security attributes that are specified in the actions that are associated with the rule.

An AT-TLS policy is matched (referred to as *policy mapping*) to a TCP connection the first time one of the following socket API calls occurs in a z/OS TCP application:

- An outbound **`connect()`** for a TCP socket.
- A **`select()`** to see whether a socket is readable or writable.
- A **`poll()`** to see whether a socket is readable or writable.
- Any call that sends or receives data over a socket.
- An **`SIOCTTLSCTL`** input/output control (IOCTL) invocation in the case of AT-TLS-aware and AT-TLS-controlling applications.

An AT-TLS rule is defined with the **`TTLSRule`** statement, which specifies a set of conditions that are compared against the TCP connection attributes. Various attributes are considered in **`TTLSRule`** conditions, including:

`LocalAddr`	Local IP address or addresses.
`RemoteAddr`	Remote IP address or addresses.
`LocalPortRange`	Local port or ports.
`RemotePortRange`	Remote port or ports.
`Jobname`	Job name of the owning application or wildcard job name.
`Userid`	UID of the owning process or wildcard UID.
`Direction`	Inbound if applied to a passive socket (established by accept), outbound if applied to an active socket (established by connect), or both.

The **`TTLSRule`** statement requires, at a minimum, the **`Direction`** condition and one other condition from the previous list. These considerations apply to rules:

- If a condition is not specified, that condition is not considered when comparing the rule and the connection for a match.
- Multiple values can be specified for the IP address and port conditions, either directly in the condition or as a referenced group.
- IPv6 addresses are fully supported.

When creating **`TTLSRules`**, you should avoid defining rules of the same kind that overlap. Example 10 shows two overlapping rules.

Example 10 Example of overlapping rules

```
TTLSRule A                                  TTLSRule B
{                                           {
  LocalAddr 1.1.1.1                           LocalAddr 1.1.1.1
  RemoteAddr 2.2.2.2                          RemoteAddr 2.2.2.2
  LocalPortRange 11000                        LocalPortRange 11000
  RemotePortRange 15000 16000                 RemotePortRange 15500 16500
  ...                                         ...
}                                           }
```

Although both rules are identical in terms of IP addresses and local ports, they define different remote port ranges that overlap with each other. Avoid such overlaps unless they are used to distinguish between policy statements for Db2 for z/OS acting as a client versus Db2 for z/OS acting as a server. When overlapping rules are necessary, you should specify a Priority value on each of the overlapping rules to tell AT-TLS the order in which to evaluate those rules relative to each other.

Priority values are integers 1 - 2,000,000,000, with 2,000,000,000 being the highest priority. When assigning priorities, skip several values between rules to allow for future rule insertion between existing rules.

Tip: If a connection can map to more than one rule, always use a priority, and leave priority space between rules.

When a rule match is found, policy lookup stops, and the connection is assigned the actions that are associated with the rule. A **TTLSRule** can reference up to three actions, where each action represents a scope of control.

Table 2 describes these AT-TLS actions.

Table 2 AT-TLS actions

Action reference	Action statement	Description
TTLSGroupActionRef	N/A	This statement is required, and it must specify **TTLSEnabled ON** or **TTLSEnabled OFF**. If **TTLSEnabled OFF** is specified, no additional action references and statements are required. If **TTLSEnabled ON** is specified, the AT-TLS environment action is required, and the AT-TLS connection action is optional.
TTLSEnvironmentActionRef	N/A	This statement is required only if the AT-TLS group action specifies **TTLSEnabled ON**. If specified, this statement requires a key ring name (either RACF or gskkyman format), and the `HandshakeRole` (either Client, Server or `ServerWithClientAuth`). Optionally, the AT-TLS connection action can be specified if a subset of connections must have separate parameters.
TTLSConnectionActionRef	**TTLSConnectionAction**	This statement is optional. It overrides **TTLSEnvironmentAction** settings for specific connections.

For more information about these AT-TLS actions, see *z/OS V2R4 Communications Server: IP Configuration Guide*, SC27-3650.

Peer authentication models

The TLS and SSL protocols support two models of peer authentication during the handshake phase, and AT-TLS and Db2 provide a couple of extra measures that build on one of those models.

The most basic model is called *server authentication*. In this model, The TLS/SSL server sends its X.509 certificate to the client so that the client can verify the server's identity. So when Db2 for z/OS is acting as a server, it sends its certificate to the client that is attempting to connect to Db2.

Server authentication is appropriate in situations where the client must ensure that it is communicating with the correct server but the server is willing to serve any client. Server authentication is configured in the AT-TLS policy by coding **HandshakeRole Server** in the **TTLSEnvironmentAction** statement.

In cases where proof of the client's identity is also important, TLS/SSL supports *client authentication* (sometimes called *mutual authentication*), which builds on server authentication. With client authentication, after the server proves its identity to the client, the client responds by sending its own X.509 certificate to the server so that the server can verify the client's identity.

Client authentication is optional, and it is appropriate in situations where the server must ensure the identity of the clients that it serves. Client authentication is configured in the AT-TLS policy by coding **`HandshakeRole ServerWithClientAuth`** on the **`TTLSEnvironmentAction`** statement. When client authentication is used, AT-TLS supports several variations regarding the actual level of client authentication through the **`ClientAuthType`** parameter of the **`TTLSEnvironmentAdvancedParms`** policy statement.

The most commonly used **`ClientAuthType`** values are:

`Required`	This is the default **`ClientAuthType`** value. It requires the client to supply its digital certificate during the TLS handshake. If no client certificate is supplied, the handshake fails. If the certificate is supplied, the handshake succeeds if System SSL successfully authenticates the certificate. To successfully authenticate the client certificate, the server's key ring must contain a trusted signing certificate that can be used to verify the client certificate's authenticity. We describe different certificate configurations in more detail later.
`SAFCheck`	Adds z/OS -specific checking to the **`ClientAuthType Required`** behavior. After System SSL successfully authenticates the client certificate, AT-TLS queries RACF to ensure that the client certificate is associated with a valid z/OS UID in RACF before allowing a secure connection to be established. If the certificate is associated with a z/OS UID, the connection is enabled. If not, the connection is rejected.

When **`ClientAuthType SAFCheck`** is specified for IBM Db2 Distributed Relational Database Architecture (IBM DRDA) connections, Db2 can add more authentication measures. After the TLS/SSL handshake and the additional AT-TLS `SAFCheck` authentication succeed, a DRDA handshake flows over the secured connection. This DRDA handshake might or might not contain user credentials. In some cases, depending on the credentials sent (or not sent) during the DRDA handshake, Db2 might query AT-TLS for the z/OS UID that is associated with the client certificate.

The returned z/OS UID is checked to ensure that it has permission to access the relevant *`d2sn`*`.DIST` resource in the RACF DSRN resource class (where *d2sn* is the Db2 subsystem name to which the client is connecting). If the z/OS UID does not have permission to access the resource, the DRDA connection is dropped. This Db2 feature, called *client certificate authentication*, enables you to restrict remote client access to the Db2 subsystem based on the client certificate mapping.

For more information about Db2 client authentication logic, see "Processing client access by using digital certificates at a Db2 for z/OS server" on page 63.

Table 3 summarizes the Db2 client authentication options available with AT-TLS when the policy covering the DRDA connection specifies HandshakeRole ServerWithClientAuth.

Table 3 Client authentication types for Db2 DRDA connections

AT-TLS ClientAuthType	Client certificate	DSRN class is active and the Db2 subsystem's profile is defined	Certificate validation	Notes
Passthru	Optional	N/A	None.	Not recommended.
Full	Optional	N/A	If a client certificate is provided, it is validated against the server's key ring. If not, the connection is enabled.	Not recommended for Db2 DRDA connections.
Required	Required	N/A	The certificate is validated against the server's key ring.	The default, which is advisable when extra z/OS UID-based access controls are not required.
SAFCheck	Required	Required	The certificate is validated against the server's key ring and must be associated with a UID in the security product. In addition, Db2 adds client certificate authentication in certain cases depending on the user credentials that are specified in the DRDA handshake.	This level is the strongest level of authentication. For more information, see "Processing client access by using digital certificates at a Db2 for z/OS server" on page 63.

Coding the AT-TLS policy rules for TLS/SSL server authentication

Because it is common for a single z/OS system to support multiple Db2 for z/OS subsystems, we illustrate the AT-TLS configuration and associated digital certificate configuration by using a scenario that supports two similar Db2 for z/OS subsystems: One is named DB2A, and the other is named DB2B. The policies are built by using a method that makes it easy to add Db2 for z/OS subsystems.

Each Db2 for z/OS subsystem is assumed to run under its own dedicated z/OS UID. Your environment might be simpler (a single Db2 subsystem) or more complex (multiple subsystems, or scenarios where a subsystem must act as both a server and a client), but you can use the same approach regardless of the number of Db2 for z/OS subsystems.

Example 11 shows a simple AT-TLS policy that protects connections to our DB2A and DB2B subsystems by using TLSv1.2 and TLSv1.3, strong cipher suites, and strong signature algorithms.

Example 11 Simple AT-TLS policy for two similar Db2 for z/OS servers to support TLS/SSL connections

```
TTLSRule DB2ASecureServer
{
   LocalPortRange 4801
   JobName DB2ADIST
   Direction Inbound
   Priority 1
   TTLSGroupActionRef DB2@GrpAct
   TTLSEnvironmentActionRef DB2@SecureServerEnvAct
}
TTLSRule DB2BSecureServer
{
   LocalPortRange 4802
   JobName DB2BDIST
   Direction Inbound
   Priority 1
   TTLSGroupActionRef DB2@GrpAct
   TTLSEnvironmentActionRef DB2@SecureServerEnvAct
}
TTLSGroupAction DB2@GrpAct
{
   TTLSEnabled On
   FIPS140 Off
   Trace 15
}
TTLSEnvironmentAction DB2@SecureServerEnvAct
{
   HandShakeRole Server
   TTLSKeyRingParmsRef DB2@KeyRing
   TTLSEnvironmentAdvancedParmsRef DB2@EnvAdvParms
   TTLSCipherParmsRef DB2@CipherParms
   TTLSSignatureParmsRef DB2@SigParms
}
TTLSKeyRingParms DB2@KeyRing
{
   Keyring DB2@KEYRING
}
TTLSEnvironmentAdvancedParms DB2@EnvAdvParms
{
  TLSv1.3  On
  TLSv1.2  On
  TLSv1.1  Off
  TLSv1    Off
  SSLv3    Off
  SSLv2    Off
}
TTLSCipherParms DB2@CipherParms
{
# TLSv1.2: ECDSA or RSA peer auth with ephemeral DH, AES-GCM, and SHA-2
  V3CipherSuites TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
  V3CipherSuites TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
  V3CipherSuites TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
  V3CipherSuites TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  V3CipherSuites TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  V3CipherSuites TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
  V3CipherSuites TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
# TLSv1.3: Ephemeral DH, AES-GCM, or ChaCha-Poly1305 and SHA-2
  V3CipherSuites TLS_AES_256_GCM_SHA384
  V3CipherSuites TLS_AES_128_GCM_SHA256
  V3CipherSuites TLS_CHACHA20_POLY1305_SHA256
}
TTLSSignatureParms DB2@SigParms
```

```
{
# Allow digital signatures with hashes of 256 bits or more
  SignaturePairs TLS_SIGALG_SHA512_WITH_RSASSA_PSS
  SignaturePairs TLS_SIGALG_SHA512_WITH_ECDSA
  SignaturePairs TLS_SIGALG_SHA512_WITH_RSA
  SignaturePairs TLS_SIGALG_SHA384_WITH_RSASSA_PSS
  SignaturePairs TLS_SIGALG_SHA384_WITH_ECDSA
  SignaturePairs TLS_SIGALG_SHA384_WITH_RSA
  SignaturePairs TLS_SIGALG_SHA256_WITH_RSASSA_PSS
  SignaturePairs TLS_SIGALG_SHA256_WITH_ECDSA
  SignaturePairs TLS_SIGALG_SHA256_WITH_RSA
}
```

Tip: TLSv1.3 and the RSA Signature Scheme with Appendix - Probabilistic Signature Scheme (RSASSA-PSS) signature algorithm are supported beginning with z/OS V2R4. If you are working with a z/OS version earlier than V2R4, you must remove the following specifications from the example:

- The TLSv1.3 protocol specification from the **TTLSEnvironmentAdvancedParms** statement.
- The TLSv1.3 cipher suites from the **TTLSCipherParms** statement.
- All signature pairs containing the RSASSA-PSS algorithm from the **TTLSSignatureParms** statement.

The two **TTLSRule** statements define the rules that protect the Distributed Data Facility (DDF) traffic. Each rule specifies three conditions that are matched against new TCP connections:

1. The connection is initiated by a remote peer to z/OS (it is an inbound connection).
2. The connection arrives on the respective local port (4801 and 4802).
3. The ports are owned by the Db2 DDF address spaces. DB2ADIST owns 4801 and DB2BDIST owns 4802.

Each of these rules reference (and share) a common set of extra policy statements, which we describe here.

The **TTLSGroupActionRef** parameter refers to the **TTLSGroupAction** statement that is named `DB2@SecureGrpAct`, which is shared by both **TTLSRule** statements. This statement provides a System SSL instance, enables AT-TLS protection for the specified TCP traffic, and specifies a trace level of 15. This trace level enables the tracing of instances when a connection is mapped to an AT-TLS rule, when a secure connection is successfully initiated, and for major AT-TLS events.

Because **TTLSEnabled ON** is specified in the `DB2@SecureGrpAct` statement, a **TTLSEnvironmentAction** statement, `DB2@SecureServerEnvAct`, is also required. The environment action statement specifies `HandshakeRole` of `Server`, which implies that z/OS acts as the server during the TLS/SSL handshake, and that the server authentication model of peer authentication is used.

The **TTLSEnvironmentAction** statement also includes a reference to a **TTLSKeyringParms** statement that defines a RACF key ring name, `DB2@KEYRING` (key ring definitions are described in “Creating and configuring digital certificates” on page 31). Each Db2 for z/OS DDF address space requires its own key ring, which is qualified by its UID. Because the AT-TLS policy lists only the key ring name without a qualifying UID, it can be used by multiple Db2 for z/OS subsystems. The only requirement is that a key ring of the same name must be created for each Db2 for z/OS subsystem’s UID. As such, DB2A uses `DB2AUSER/DB2@KEYRING` and DB2B uses `DB2BUSER/DB2@KEYRING` at run time.

The **TTLSEnvironmentAction** also refers to the **TTLSEnvironmentAdvancedParms** statement that lists which TLS protocol versions are acceptable. In this case, we allow only TLSv1.2 and TLSv1.3 because the older protocol versions have weaknesses. At the time of writing, TLSv1.2 and TLSv1.3 are generally accepted as providing strong TLS security, and all the client implementations that are described in this paper support at least TLSv1.2. However, if you are working with older Db2 client software, you might need to modify this list to match the capabilities of your communications partners. For a TLS/SSL handshake to succeed, the server and client must support at least one TLS protocol version in common, so you might need to customize your server's list of supported TLS protocols to include one that your client supports.

TTLSEnvironmentAction also refers to the **TTLSCipherParms** statement that lists the cipher suites that are allowed. In this case, we allow only cipher suites that use strong algorithms (at the time of writing) for peer authentication, key exchange, symmetric encryption, and message authentication. The **TTLSCipherParms** statement specifies the list of acceptable TLS/SSL cipher suites in order of preference (top to bottom, most preferred to least).

If you do not specify a **TTLSCipherParms** statement, AT-TLS uses System SSL's default cipher list. In most cases (when z/OS Cryptographic Services Security Level 3 feature is installed), that list contains the following ciphers in the order that is shown:

```
(35) TLS_RSA_WITH_AES_256_CBC_SHA
(38) TLS_DHE_DSS_WITH_AES_256_CBC_SHA
(39) TLS_DHE_RSA_WITH_AES_256_CBC_SHA
(2F) TLS_RSA_WITH_AES_128_CBC_SHA
(32) TLS_DHE_DSS_WITH_AES_128_CBC_SHA
(33) TLS_DHE_RSA_WITH_AES_128_CBC_SHA
```

There are a few important points to note here:

- This default list does not contain the strongest suites that are available (the ones that are using ephemeral elliptic curve Diffie-Hellman, Advanced Encryption Standard (AES)-GCM, and SHA2-based message authentication), so we advise specifying a **TTLSCipherParms** statement that enforces a minimum cryptographic strength that meets your company's security policies.
- If you must comply with the US National Institute of Standards and Technology (NIST) Special Publication SP800-131a, you must code a **TTLSCipherParms** statement that includes only compliant cipher suites (the one that is shown in Example 11 on page 18 complies with NIST SP800-131a, with preference given to the strongest suites).
- If you must comply with NIST Special Publication SP800-52 as part of compliance with Federal Information Processing Standards Security Requirements for Cryptographic Modules (FIPS 140-2), you must change **TTLSGroupAction** from `FIPS140 Off` to `FIPS140 On`. This change ensures that Db2 for z/OS use only cipher suites that comply with FIPS140-2 requirements.

Another reason that you might need to specify the list of cipher suites is to match the capabilities of your communications partners. For a TLS/SSL handshake to succeed, the server and client must support at least one cipher suite in common, so you might need to customize your server's cipher suite list to include a cipher that your client supports.

TTLSEnvironmentAction also refers to the **TTLSSignatureParms** statement that lists which pairs of digital signature algorithms are acceptable. TLSv1.2 and TLSv1.3 both use this list to determine whether the digital signatures that are used to sign the server, client (in the case of client authentication), and certificate authority (CA) certificates that are used for peer authentication are acceptable. TLSv1.3 also uses these pairs to determine how to sign some of the TLS handshake messages.

In our case, we allow only signatures that use Elliptic Curve Digital Signature Algorithm (ECDSA), RSASSA-PSS, and Rivest-Shamir-Adleman (RSA) signatures with at least 256-bit hashes, which conform with generally accepted practices at the time of writing. However, if you are working with Db2, whose digital certificates were signed with shorter hashes or the DSA algorithm, you might need to modify this list to match the capabilities of your communications partners.

There are a few other AT-TLS policy statements that you should be aware of that we do not expand upon in this document:

`TTLSConnectionAction`

This action is optional and needed only when a subset of connections in an AT-TLS environment requires different parameters. **`TTLSConnectionAction`** serves the same purpose as **`TTLSEnvironmentAction`**, but for a specific connection. When a **`TTLSConnectionAction`** statement is associated with **`TTLSRule`**, its settings override the setting in the **`TTLSEnvironmentAction`** statement.

`TTLSConnectionAdvancedParms`

This statement is also optional. **`TTLSConnectionAdvancedParms`** serves the same purpose as the **`TTLSEnvironmentAdvancedParms`** statement, but for a specific connection. If one of these statements is associated with **`TTLSConnectionAction`**, its settings override the setting in the associated **`TTLSEnvironmentAdvancedParms`** statement.

`TTLSGskAdvancedParms`

Specifies advanced attributes that are specific to System SSL. The **`TTLSGskAdvancedParms`** statement can be contained in or referenced by **`TTLSEnvironmentAction`** statements.

`TTLSGskOcspParms`, `TTLSGskHttpCdpParms`, and `TTLSGskLdapParms`

Specifies parameters that are related to the different certificate revocation mechanisms that are supported by System SSL. These statements can be contained in or referenced by **`TTLSEnvironmentAction`** statements.

For more information about AT-TLS policy configuration, see Chapter 20, “Application Transparent Transport Layer Security data protection”, in *z/OS V2R4 Communications Server: IP Configuration Guide*, SC27-3650.

For the detailed syntax of each AT-TLS policy statement, see Chapter 16, “Policy Agent and policy applications”, in *z/OS V2R4 Communications Server: IP Configuration Reference*, SC27-3651.

Coding the AT-TLS policy rules for TLS/SSL client authentication

If you must verify the identity of each client that connects to Db2, use TLS/SSL client authentication. To do this task, specify the **`HandshakeRole ServerWithClientAuth`** parameter for the **`TTLSEnvironmentAction`** statement in the AT-TLS policy, as shown in Example 12.

Example 12   AT-TLS server policy for TLS/SSL client authentication

```
.
.
.
TTLSEnvironmentAction DB2@SecureServerEnvAct
{
   HandShakeRole ServerWithClientAuth
   TTLSKeyRingParmsRef DB2@KeyRing
   TTLSEnvironmentAdvancedParms DB2@EnvAdvParms
   TTLSCipherParmsRef DB2@CipherParms
   TTLSSignatureParmsRef DB2@SigParms
}
.
.
.
TTLSEnvironmentAdvancedParms DB2@EnvAdvParms
{
   ClientAuthType SAFCheck
   TLSv1.3  On
   TLSv1.2  On
   TLSv1.1  Off
   TLSv1    Off
   SSLv3    Off
   SSLv2    Off
}
.
.
.
```

Example 12 on page 22 specifies the **HandshakeRole** of **ServerWithClientAuth**, and adds a parameter to the **TTLSEnvironmentAdvancedParms** statement to specify a client authentication type (`ClientAuthType`) of `SAFCheck`. This level of client authentication requires the client certificate identity to be associated with a z/OS UID in RACF (see “Registering a client certificate with RACF (optional)” on page 40).

Additionally, depending on how your Db2 subsystem obtains the client user credentials, this UID might also need to be given permission to access the relevant *d2sn*.`DIST` resource in RACF, as described in “Peer authentication models” on page 16.

Refreshing policies

After you define the AT-TLS rules, if PAGENT is started, you must update or refresh the PAGENT before the AT-TLS rules take effect. The **MVS MODIFY** command is used to refresh or update the PAGENT. These commands use the following syntax:

F PAGENT,UPDATE The **UPDATE** command causes the PAGENT to reread and process the policy files. With **UPDATE**, PAGENT installs or removes from the stack any new, changed, or deleted policies (any unchanged policies are unaffected). Therefore, we advise that you use this command in most cases.

F PAGENT,REFRESH The **REFRESH** command causes the PAGENT to reread and process the policy files. If the **FLUSH** parameter was specified on the **TcpImage** configuration statement, the **REFRESH** command triggers **FLUSH** processing.

Because **FLUSH** deletes and reinstalls all policies, you should use this command only if you suspect that policies have somehow become out of sync between the TCP/IP stack and the PAGENT. One consequence of triggering **FLUSH** processing is that policy statistics that are collected in the TCP/IP stack are reset.

Verifying policies

Use the z/OS UNIX **pasearch** command to query information from the PAGENT. The command can be issued from the UNIX Systems Services shell or from the Time Sharing Option Extensions (TSO/E) **oshell** command. The **pasearch** output in Example 13 was produced by issuing the TSO/E **oshell** command to run the **pasearch** command specifying the **-t** option to display all AT-TLS policy entries:

```
oshell pasearch -t
```

Example 13 Displaying AT-TLS policies by using the pasearch -t command

```
TCP/IP pasearch CS V2R4                   Image Name: TCPCS
  Date:                 06/23/2021        Time:  13:56:36
  TTLS Instance ID:     1624470892

policyRule:             DB2ASecureServer
  Rule Type:            TTLS
  Version:              3                 Status:           Active
  Weight:               1                 ForLoadDist:      False
  Priority:             1                 Sequence Actions:  Don't Care
  No. Policy Action:    2
  policyAction:         DB2@GrpAct
   ActionType:          TTLS Group
   Action Sequence:     0
  policyAction:         DB2@SecureServerEnvAct
   ActionType:          TTLS Environment
   Action Sequence:     0
  Time Periods:
   Day of Month Mask:
   First to Last:       1111111111111111111111111111111
   Last to First:       1111111111111111111111111111111
   Month of Yr Mask:    111111111111
   Day of Week Mask:    1111111  (Sunday - Saturday)
   Start Date Time:     None
   End Date Time:       None
   Fr TimeOfDay:        00:00             To TimeOfDay:      24:00
   Fr TimeOfDay UTC:    04:00             To TimeOfDay UTC:  04:00
   TimeZone:            Local
  TTLS Condition Summary:                 NegativeIndicator: Off
   Local Address:
    FromAddr:           All
    ToAddr:             All
   Remote Address:
    FromAddr:           All
    ToAddr:             All
   LocalPortFrom:       4801              LocalPortTo:       4801
   RemotePortFrom:      0                 RemotePortTo:      0
   JobName:             DB2ADIST          UserId:
   ServiceDirection:    Inbound
  Policy created: Wed Jun 23 13:54:52 2021
  Policy updated: Wed Jun 23 13:54:52 2021

  TTLS Action:                  DB2@GrpAct
    Version:                    3
    Status:                     Active
    Scope:                      Group
    TTLSEnabled:                On
    CtraceClearText:            Off
    Trace:                      15
    FIPS140:                    Off
    TTLSGroupAdvancedParms:
     SecondaryMap:              Off
     SyslogFacility:            Daemon
    Policy created: Wed Jun 23 13:54:52 2021
    Policy updated: Wed Jun 23 13:54:52 2021

  TTLS Action:                  DB2@SecureServerEnvAct
    Version:                    3
    Status:                     Active
    Scope:                      Environment
    HandshakeRole:              Server
    SuiteBProfile:              Off
    TTLSKeyringParms:
     Keyring:                   DB2@KEYRING
    TTLSEnvironmentAdvancedParms:
     SSLv2:                     Off
     SSLv3:                     Off
     TLSv1:                     Off
     TLSv1.1:                   Off
     TLSv1.2:                   On
     TLSv1.3:                   On
     MiddleBoxCompatMode:       Off
     ApplicationControlled:     Off
     HandshakeTimeout:          10
     ClientAuthType:            Required
     ResetCipherTimer:          0
     TruncatedHMAC:             Off
     CertValidationMode:        Any
     ServerMaxSSLFragment:      Off
     ClientMaxSSLFragment:      Off
     ServerHandshakeSNI:        Off
     ClientHandshakeSNI:        Off
     Renegotiation:             Default
     RenegotiationIndicator:    Optional
     RenegotiationCertCheck:    Off
     3DesKeyCheck:              Off
     ClientEDHGroupSize:        Legacy
     ServerEDHGroupSize:        Legacy
     PeerMinCertVersion:        Any
     PeerMinDHKeySize:          1024
     PeerMinDsaKeySize:         1024
     PeerMinECCKeySize:         192
     PeerMinRsaKeySize:         1024
     ServerScsv:                Off
    TTLSSignatureParms:
     ClientECurves:
      0021  secp224r1
      0023  secp256r1
      0024  secp384r1
      0025  secp521r1
      0019  secp192r1
      0029  X25519
     ClientKeyShareGroups:
      0023  secp256r1
     ServerKeyShareGroups:
      0023  secp256r1
      0024  secp384r1
      0025  secp521r1
      0029  X25519
      0030  X448
```

```
        SignaturePairs:
         0806  TLS_SIGALG_SHA512_WITH_RSASSA_PSS
         0603  TLS_SIGALG_SHA512_WITH_ECDSA
         0601  TLS_SIGALG_SHA512_WITH_RSA
         0805  TLS_SIGALG_SHA384_WITH_RSASSA_PSS
         0503  TLS_SIGALG_SHA384_WITH_ECDSA
         0501  TLS_SIGALG_SHA384_WITH_RSA
         0804  TLS_SIGALG_SHA256_WITH_RSASSA_PSS
         0403  TLS_SIGALG_SHA256_WITH_ECDSA
         0401  TLS_SIGALG_SHA256_WITH_RSA
       TTLSCipherParms:
        v3CipherSuites:
         C02C  TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
         C030  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
         C02B  TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
         C02F  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
         009F  TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
         009E  TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
         1302  TLS_AES_256_GCM_SHA384
         1301  TLS_AES_128_GCM_SHA256
         1303  TLS_CHACHA20_POLY1305_SHA256
       TTLSGskAdvancedParms:
        GSK_V3_SESSION_TIMEOUT:                    86400
        GSK_V3_SIDCACHE_SIZE:                      512
        GSK_SESSION_TICKET_CLIENT_ENABLE:          On
        GSK_SESSION_TICKET_CLIENT_MAXSIZE:         8192
        GSK_SESSION_TICKET_SERVER_ENABLE:          On
        GSK_SESSION_TICKET_SERVER_ALGORITHM:       AESCBC128
        GSK_SESSION_TICKET_SERVER_COUNT:           2
        GSK_SESSION_TICKET_SERVER_KEY_REFRESH:     300
        GSK_SESSION_TICKET_SERVER_TIMEOUT:         300
        TTLSGskHttpCdpParms:
         HttpCdpEnable:            Off
         HttpCdpProxyServerPort:   80
         HttpCdpResponseTimeout:   15
         HttpCdpMaxResponseSize:   204800
         HttpCdpCacheSize:         32
         HttpCdpCacheEntryMaxsize: 0
        TTLSGskOcspParms:
         OcspAiaEnable:            Off
         OcspProxyServerPort:      80
         OcspRetrieveViaGet:       Off
         OcspUrlPriority:          On
         OcspRequestSigalg:
          0401  TLS_SIGALG_SHA256_WITH_RSA
         OcspClientCacheSize:      256
         OcspCliCacheEntryMaxsize: 0
         OcspNonceGenEnable:       Off
         OcspNonceCheckEnable:     Off
         OcspNonceSize:            8
         OcspResponseTimeout:      15
         OcspMaxResponseSize:      20480
         OcspServerStapling:       Off
       Policy created: Wed Jun 23 13:54:52 2021
       Policy updated: Wed Jun 23 13:54:52 2021

   policyRule:             DB2BSecureServer
     Rule Type:            TTLS
     Version:              3                 Status:           Active
     Weight:               1                 ForLoadDist:      False
```

```
Priority:             1                 Sequence Actions:  Don't Care
No. Policy Action:    2
policyAction:         DB2@GrpAct
 ActionType:          TTLS Group
 Action Sequence:     0
policyAction:         DB2@SecureServerEnvAct
 ActionType:          TTLS Environment
 Action Sequence:     0
Time Periods:
 Day of Month Mask:
 First to Last:       1111111111111111111111111111111
 Last to First:       1111111111111111111111111111111
 Month of Yr Mask:    111111111111
 Day of Week Mask:    1111111  (Sunday - Saturday)
 Start Date Time:     None
 End Date Time:       None
 Fr TimeOfDay:        00:00             To TimeOfDay:      24:00
 Fr TimeOfDay UTC:    04:00             To TimeOfDay UTC:  04:00
 TimeZone:            Local
TTLS Condition Summary:                 NegativeIndicator: Off
 Local Address:
  FromAddr:           All
  ToAddr:             All
 Remote Address:
  FromAddr:           All
  ToAddr:             All
 LocalPortFrom:       4802              LocalPortTo:       4802
 RemotePortFrom:      0                 RemotePortTo:      0
 JobName:             DB2BDIST          UserId:
 ServiceDirection:    Inbound
Policy created: Wed Jun 23 13:54:52 2021
Policy updated: Wed Jun 23 13:54:52 2021

TTLS Action:                  DB2@GrpAct
  Version:                    3
  Status:                     Active
  Scope:                      Group
  TTLSEnabled:                On
  CtraceClearText:            Off
  Trace:                      15
  FIPS140:                    Off
  TTLSGroupAdvancedParms:
   SecondaryMap:              Off
   SyslogFacility:            Daemon
  Policy created: Wed Jun 23 13:54:52 2021
  Policy updated: Wed Jun 23 13:54:52 2021

TTLS Action:                  DB2@SecureServerEnvAct
  Version:                    3
  Status:                     Active
  Scope:                      Environment
  HandshakeRole:              Server
  SuiteBProfile:              Off
  TTLSKeyringParms:
   Keyring:                   DB2@KEYRING
  TTLSEnvironmentAdvancedParms:
   SSLv2:                     Off
   SSLv3:                     Off
   TLSv1:                     Off
   TLSv1.1:                   Off
   TLSv1.2:                   On
   TLSv1.3:                   On
   MiddleBoxCompatMode:       Off
   ApplicationControlled:     Off
   HandshakeTimeout:          10
   ClientAuthType:            Required
   ResetCipherTimer:          0
   TruncatedHMAC:             Off
   CertValidationMode:        Any
   ServerMaxSSLFragment:      Off
   ClientMaxSSLFragment:      Off
   ServerHandshakeSNI:        Off
   ClientHandshakeSNI:        Off
   Renegotiation:             Default
   RenegotiationIndicator:    Optional
   RenegotiationCertCheck:    Off
   3DesKeyCheck:              Off
   ClientEDHGroupSize:        Legacy
   ServerEDHGroupSize:        Legacy
   PeerMinCertVersion:        Any
   PeerMinDHKeySize:          1024
   PeerMinDsaKeySize:         1024
   PeerMinECCKeySize:         192
   PeerMinRsaKeySize:         1024
   ServerScsv:                Off
  TTLSSignatureParms:
   ClientECurves:
    0021  secp224r1
    0023  secp256r1
    0024  secp384r1
    0025  secp521r1
    0019  secp192r1
    0029  X25519
   ClientKeyShareGroups:
    0023  secp256r1
   ServerKeyShareGroups:
    0023  secp256r1
    0024  secp384r1
    0025  secp521r1
    0029  X25519
    0030  X448
   SignaturePairs:
    0806  TLS_SIGALG_SHA512_WITH_RSASSA_PSS
    0603  TLS_SIGALG_SHA512_WITH_ECDSA
    0601  TLS_SIGALG_SHA512_WITH_RSA
    0805  TLS_SIGALG_SHA384_WITH_RSASSA_PSS
    0503  TLS_SIGALG_SHA384_WITH_ECDSA
    0501  TLS_SIGALG_SHA384_WITH_RSA
    0804  TLS_SIGALG_SHA256_WITH_RSASSA_PSS
    0403  TLS_SIGALG_SHA256_WITH_ECDSA
    0401  TLS_SIGALG_SHA256_WITH_RSA
  TTLSCipherParms:
   v3CipherSuites:
    C02C  TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
    C030  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    C02B  TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    C02F  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    009F  TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
    009E  TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
    1302  TLS_AES_256_GCM_SHA384
```

```
    1301  TLS_AES_128_GCM_SHA256
    1303  TLS_CHACHA20_POLY1305_SHA256
  TTLSGskAdvancedParms:
   GSK_V3_SESSION_TIMEOUT:                   86400
   GSK_V3_SIDCACHE_SIZE:                     512
   GSK_SESSION_TICKET_CLIENT_ENABLE:         On
   GSK_SESSION_TICKET_CLIENT_MAXSIZE:        8192
   GSK_SESSION_TICKET_SERVER_ENABLE:         On
   GSK_SESSION_TICKET_SERVER_ALGORITHM:      AESCBC128
   GSK_SESSION_TICKET_SERVER_COUNT:          2
   GSK_SESSION_TICKET_SERVER_KEY_REFRESH:    300
   GSK_SESSION_TICKET_SERVER_TIMEOUT:        300
   TTLSGskHttpCdpParms:
    HttpCdpEnable:           Off
    HttpCdpProxyServerPort:  80
    HttpCdpResponseTimeout:  15
    HttpCdpMaxResponseSize:  204800
    HttpCdpCacheSize:        32
    HttpCdpCacheEntryMaxsize: 0
   TTLSGskOcspParms:
    OcspAiaEnable:           Off
    OcspProxyServerPort:     80
    OcspRetrieveViaGet:      Off
    OcspUrlPriority:         On
    OcspRequestSigalg:
     0401  TLS_SIGALG_SHA256_WITH_RSA
    OcspClientCacheSize:     256
    OcspCliCacheEntryMaxsize: 0
    OcspNonceGenEnable:      Off
    OcspNonceCheckEnable:    Off
    OcspNonceSize:           8
    OcspResponseTimeout:     15
    OcspMaxResponseSize:     20480
    OcspServerStapling:      Off
  Policy created: Wed Jun 23 13:54:52 2021
  Policy updated: Wed Jun 23 13:54:52 2021
```

Diagnosing AT-TLS problems

There are several useful tools for diagnosing AT-TLS problems. For a description of common AT-TLS start-up errors, debugging steps and tools, and descriptions of AT-TLS return codes, see Chapter 27, “Diagnosing Application Transparent Transport Layer Security (AT-TLS)”, in *z/OS V2R4 Communications Server IP Diagnosis Guide*, GC27-3652. That document explains how to diagnose TCP/IP problems and determine whether a specific problem is in the z/OS Communications Server TCP/IP component. The document also explains how to gather information for and describe problems to the IBM Software Support Center.

For your convenience, here is a summary of the tools that you can use to diagnose AT-TLS problems in the order that you would usually use them:

- Commands:
  - The **`pasearch`** command provides details about the AT-TLS policies that are installed.
  - The TSO **`Netstat`** command and its z/OS UNIX counterpart **`netstat`** provide AT-TLS information as it relates to specific TCP connections. The **`COnn/-c`** and **`TTLS/-x`** options are especially useful for AT-TLS diagnosis.

  For a complete description of these commands and their output, see *z/OS V2R4 Communications Server IP System Administrator’s Commands*, SC27-3661.

- Any errors that are detected by the policy agent when parsing the AT-TLS policy statements are written to the PAGENT log, which can be either a z/OS UNIX file or `syslogd`.

  The specific PAGENT log destination is controlled by the PAGENT **`-L`** startup parameter. Either **`SYSLOGD`** or a z/OS UNIX file can be specified. If **`SYSLOGD`** is specified, `syslogd` takes the messages and writes them under the facility name daemon to a destination (typically a z/OS UNIX file) that is specified in the `/etc/syslog.conf` file. The environment variable **`PAGENT_LOG_FILE`** also specifies the destination of the log file by using the same format as this option. The **`-L`** option overrides the **`PAGENT_LOG_FILE`** environment variable.

  If the PAGENT log file destination is not specified by either the **`-L`** parameter or the **`PAGENT_LOG_FILE`** environment variable, the default location is `/tmp/pagent.log`. For more information about specifying the PAGENT log file destination, see "Starting Policy Agent from the z/OS shell" in Chapter 16, "Policy Agent and policy applications", in *z/OS V2R4 Communications Server: IP Configuration Reference*, SC27-3651.

- AT-TLS messages are written to `syslogd` under the facility name daemon. `Syslogd` takes the messages and writes them to a destination (typically a z/OS UNIX file) that is specified in the `/etc/syslog.conf` file.

  AT-TLS messages begin with the prefix `EZD`. Most are in the range `EZD1281I` - `EZD1290I`. Descriptions of the messages and suggested actions can be found in *z/OS V2R4 Communications Server: IP Messages Volume 2 (EZB, EZD)*, SC27-3655.

  AT-TLS error messages contain detailed return codes that are useful in isolating problems. The following convention is used for AT-TLS return code values:

  – Return code values 1 - 5000 come directly from System SSL. These return codes, including detailed descriptions and suggested actions, are described in Chapter 13, "Messages and codes", in *z/OS V2R4 Cryptographic Services System Secure Sockets Layer Programming*, SC14-7495.

  – Return code values 5001 and higher are generated by AT-TLS code in the TCP/IP stack. These return codes, including detailed descriptions and suggested actions, are described in Chapter 27, "Diagnosing Application Transparent Transport Layer Security (AT-TLS)", in *z/OS V2R4 Communications Server IP Diagnosis Guide*, GC27-3652.

  For more information about configuring `syslogd`, see the "Configuring the syslog daemon" topic in Chapter 5, "TCP/IP Customization", in *z/OS V2R4 Communications Server: IP Configuration Guide*, SC27-3650.

- AT-TLS traces can be enabled through the AT-TLS policy **`Trace`** parameter of the **`TTLSEnvironmentAction`** or TTLS **`ConnectionAction`** statements. AT-TLS trace records are also written to `syslogd` under the facility name daemon. Depending on the level of tracing that you choose, these trace records can provide many details of the internal AT-TLS operation that can help you isolate errors.

- System SSL traces. In some cases, you might need to examine the details of System SSL's operation regarding a problem. To do this task, enable System SSL tracing. For more information about setting up and using System SSL traces, see Chapter 12, "Obtaining diagnostic information", in *z/OS V2R4 Cryptographic Services System Secure Sockets Layer Programming*, SC14-7495. When using AT-TLS, *you must enable System SSL trace through the z/OS component trace facility*. AT-TLS does not support enabling System SSL trace through environment variables.

Example 14 on page 31 shows a Db2 message that is often generated in the syslog when a DRDA connection cannot be established because of an AT-TLS configuration error. In this case, the error causes Db2 to fail while attempting to process a TLS/SSL message.

Example 14 Network policy issue

```
DSNL032I -DB2R DSNLIRTR DRDA EXCEPTION CONDITION IN
REQUEST FROM REQUESTOR LOCATION=::10.0.45.11 FOR THREAD WITH
LUWID=GA002D0B.PB61.CD2843B2841D
REASON=00D3101A
ERROR ID=DSNLIRTR0003
IFCID=0192
SEE TRACE RECORD WITH IFCID SEQUENCE NUMBER=00000008
```

The following message is the corresponding client message that is generated in `db2diaglog` on a non-Java client:

```
DIA3604E The SSL function "gsk_secure_soc_init" failed with the return code "410"
in "sqlccSSLSocketSetup".
```

For more information about the possible IBM Global Security Kit for SSL (IBM GSKit) return codes, see the following website:

https://www.ibm.com/docs/en/txseries/8.1.0?topic=communications-gskit-return-codes

Creating and configuring digital certificates

There are numerous ways to create and manage digital certificates and their related key pairs. On z/OS, these ways include the following options:

- z/OS Security Server RACF (and other SAF-compliant external security managers).
- The **`gskkyman`** tool that is included with System SSL.
- z/OS Cryptographic Services PKI Services.

The **`gskkyman`** tool and RACF are suited for smaller scale certificate deployments, and public key infrastructure services (PKI Services) is a full-blown CA and management system that can manage digital certificates on an enterprise level.

There are many facilities on other platforms from CLI tools to complete digital certificate management systems (CMSs) that can be used too. When these tools are used, the certificates and keys they generate must be imported onto z/OS by using either RACF or **`gskkyman`**.

If your company has an established PKI, the keys and certificates that you use for your secure Db2 connections most likely come from that infrastructure. However, when you are experimenting with the TLS/SSL protection of your Db2 connections, you probably want to experiment with more localized tools. Likewise, if you do not have an established PKI, you might choose to use a more localized facility to create and manage your certificates and keys.

In this paper, we use RACF exclusively to generate and manage certificates and keys in SAF *key rings* on z/OS. We also show examples of a few different non- z/OS utilities that are commonly used in some of the Db2 client environments.

For more information about **`gskkyman`**, see Chapter 10, “Certificate/Key management”, in *z/OS V2R4 Cryptographic Services System Secure Sockets Layer Programming*, SC14-7495. For more information about z/OS PKI Services, see *z/OS V2R4 Cryptographic Services PKI Services Guide and Reference*, SA23-2286.

You can use RACF to perform the following tasks regarding X.509 digital certificates on z/OS:

- Create, register, store, and administer digital certificates and their associated private keys.
- Build certificate requests that can be sent to a CA for signing.
- Create and manage key rings of stored digital certificates.

Digital certificates and key rings are managed in RACF primarily by using the **`RACDCERT`** command. In this section, we describe how to use the **`RACDCERT`** command to administer digital certificates and key rings.

For a complete description of the **`RACDCERT`** command and all the functions that it supports, see Chapter 5, "RACF command syntax", in *z/OS V2R4 Security Server RACF Command Language Reference*, SA23-2292.

The next sections describe the following topics:

- Setting up access controls for certificates and key rings
- Creating the keys, digital certificates, and key rings by using RACDCERT
- Registering a client certificate with RACF (optional)
- Creating and activating client certificate name filters

Setting up access controls for certificates and key rings

In "Coding the AT-TLS policy rules for TLS/SSL server authentication" on page 18, where we defined the AT-TLS rules DB2ASecureServer and DB2BSecureServer, one of the conditions that was specified was the JobName condition. The JobNames that were specified were DB2ADIST and DB2BDIST (the started task job names of the respective Db2 DDF address spaces), which indicates to AT-TLS that the rules apply only if a job with the specified name owns the local socket for the TCP connection.

Because AT-TLS invokes System SSL under the UID of the application that owns the TCP connection, we must grant sufficient authority to the Db2 started task UIDs to access their own RACF key rings, certificates, and private keys.

In our example, the UID that is associated with the started task DB2ADIST is DB2AUSER, and the UID that is associated with DB2BDIST is DB2BUSER.

We also assume that a separate administrative UID is used for creating and managing certificates and key rings in RACF, which includes the certificates and keys that are owned by to the two Db2 started task UIDs and the certificate authority (CERTAUTH) certificates.

For illustration purposes, we call this UID `CERTMGR`. In practice, the UID can be whatever UID is appropriate in your environment.

All three of these UIDs require appropriate permissions to their own RACF certificate-related resources.

Controlling Db2 access to key rings

There are two mechanisms for controlling an application's access to key rings at run time. The preferred approach uses the RDATALIB class and the other approach uses the FACILITY class. The RDATALIB approach is preferred because it provides granular authorization to a specified key ring. In contrast, the FACILITY class allows global authorization to all key rings and certificates. For more information, see *z/OS V2R4 Security Server RACF Callable Services*, SA23-2293.

- Controlling access by using the `RDATALIB` class (recommended):
  a. If they are not already defined, create the RDATALIB definitions that are required to control access to the RACF key rings that belong to each of our Db2 instances by issuing the following TSO commands:
  ```
  SETROPTS CLASSACT(RDATALIB) RACLIST(RDATALIB)
  RDEFINE RDATALIB DB2AUSER.DB2@KEYRING.LST UACC(NONE)
  RDEFINE RDATALIB DB2BUSER.DB2@KEYRING.LST UACC(NONE)
  ```
  b. To permit the Db2 instance UIDs to access the certificates and keys that are connected to their respective key rings, issue the following TSO commands:
  ```
  PERMIT DB2AUSER.DB2@KEYRING.LST CLASS(RDATALIB) ID(DB2AUSER) ACCESS(READ)
  PERMIT DB2BUSER.DB2@KEYRING.LST CLASS(RDATALIB) ID(DB2BUSER) ACCESS(READ)
  ```
  c. Refresh the RDATALIB class:
  ```
  SETROPTS RACLIST(RDATALIB) REFRESH
  ```
- Controlling access by using the FACILITY class:
  a. If they are not already defined, create the definitions that are required to allow certificates to be stored and accessed from the RACF database by issuing the following TSO commands:
  ```
  RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
  RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
  ```
  b. To permit the Db2 started task UIDs to access the required resources, issue the following TSO commands:
  ```
  PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(DB2AUSER) ACCESS(READ)
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(DB2AUSER) ACCESS(READ)
  PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(DB2BUSER) ACCESS(READ)
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(DB2BUSER) ACCESS(READ)
  ```
  c. Refresh the FACILITY class:
  ```
  SETROPTS RACLIST(FACILITY) REFRESH
  ```

Controlling certificate administrator access to the RACDCERT command

The **`RACDCERT`** command is your primary administrative tool for managing digital certificates and key rings in RACF. It is the command that the `CERTMGR` UID uses to perform its certificate and key ring management tasks.

In our example, the authority to use the **`RACDCERT`** command is controlled through resources that are defined in the FACILITY class. The **`RACDCERT`** command is used to manage resources in the DIGTCERT, DIGTRING, DIGTNMAP, and USER classes.

Tip: You do not have to activate the DIGTCERT and DIGTRING classes to use the resources in these classes. However, performance is improved when you activate and RACLIST these classes.

To control the use of the **RACDCERT** command, authority to the `IRR.DIGTCERT.function` resource in the FACILITY class must be granted to a user unless the user has the SPECIAL attribute. One of the following authorities must be granted to the user:

- READ access to the `IRR.DIGTCERT.function` to issue the **RACDCERT** command for themselves
- UPDATE access to the `IRR.DIGTCERT.function` to issue the **RACDCERT** command for others
- CONTROL access to the `IRR.DIGTCERT.function` to issue the **RACDCERT** command for SITE and CERTAUTH certificates (this authority also includes other uses)

In our scenario, the `CERTMGR` UID creates and manages the digital certificates, private keys, and key rings that belong to the Db2 started task UIDs and any required CERTAUTH certificates. CERTMGR must be authorized to use the **RACDCERT** command.

To define profiles and permit the `CERTMGR` UID to use the required **RACDCERT** commands, complete the following steps:

1. If they are not already defined, define the following profiles to control access to the specific **RACDCERT** commands that are needed by the `CERTMGR` UID by issuing the following TSO commands:

   ```
   RDEFINE FACILITY IRR.DIGTCERT.ADD UACC(NONE)
   RDEFINE FACILITY IRR.DIGTCERT.ADDRING UACC(NONE)
   RDEFINE FACILITY IRR.DIGTCERT.CONNECT UACC(NONE)RDEFINE FACILITY
   IRR.DIGTCERT.GENCERT UACC(NONE)
   ```

2. Issue the following TSO commands to grant the required permission to the `CERTMGR` UID:

   ```
   PERMIT IRR.DIGTCERT.ADD CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
   PERMIT IRR.DIGTCERT.ADDRING CLASS(FACILITY) ID(CERTMGR) ACC(UPDATE)
   PERMIT IRR.DIGTCERT.CONNECT CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
   PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
   PERMIT IRR.DIGTCERT.EXPORT CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
   PERMIT IRR.DIGTCERT.MAP CLASS(FACILITY) ID(CERTMGR) ACC(UPDATE)
   PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
   PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CERTMGR) ACC(UPDATE)
   ```

3. Refresh the FACILITY class:

   ```
   SETROPTS RACLIST(FACILITY) REFRESH
   ```

4. If the RDATALIB class is being used to control access to the Db2 instance key rings, the `CERTMGR` UID must also be permitted to access the RDATALIB profile for each of those key rings:

   ```
   PERMIT DB2AUSER.DB2@KEYRING.LST CLASS(RDATALIB) ID(CERTMGR) ACCESS(UPDATE)
   PERMIT DB2BUSER.DB2@KEYRING.LST CLASS(RDATALIB) ID(CERTMGR) ACCESS(UPDATE)
   SETROPTS CLASSACT(RDATALIB) REFRESH
   ```

Examples: Tying the certificate and key ring permissions together

Here are examples for setting up certificate and key ring permissions by using each of the approaches that are described in “Setting up access controls for certificates and key rings” on page 32:

- Controlling access by using the RDATALIB and FACILITY classes (recommended)

  Example 16 on page 36 illustrates how to grant the appropriate authority to our three UIDs by using the RDATALIB and FACILITY classes.

Example 15 Granting permission for RACF digital certificate resources by using the RDATALIB and FACILITY classes

```
//RACDCERT EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSUADS DD DSN=SYS1.UADS,DISP=SHR
//SYSLBC DD DSN=SYS1.BRODCAST,DISP=SHR
//SYSTSIN DD *
* Activate required SAF classes and define required resources
SETROPTS CLASSACT(RDATALIB) RACLIST(RDATALIB)
RDEFINE RDATALIB DB2AUSER.DB2@KEYRING.LST UACC(NONE)
RDEFINE RDATALIB DB2BUSER.DB2@KEYRING.LST UACC(NONE)
SETROPTS CLASSACT(DIGTCERT DIGTRNG)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
* Give Db2 for z/OS user IDs sufficient authority so System SSL can use RACF
cert APIs
PERMIT DB2AUSER.DB2@KEYRING.LST CLASS(RDATALIB) ID(DB2AUSER) ACCESS(READ)
PERMIT DB2BUSER.DB2@KEYRING.LST CLASS(RDATALIB) ID(DB2BUSER) ACCESS(READ)
* Give cert management user ID sufficient access to manage certificates and key
rings
* on behalf of Db2 user IDs and for certificate authorities
PERMIT IRR.DIGTCERT.ADD CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
PERMIT IRR.DIGTCERT.ADDRING CLASS(FACILITY) ID(CERTMGR) ACC(UPDATE)
PERMIT IRR.DIGTCERT.CONNECT CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
PERMIT IRR.DIGTCERT.EXPORT CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
PERMIT IRR.DIGTCERT.MAP CLASS(FACILITY) ID(CERTMGR) ACC(UPDATE)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CERTMGR) ACC(UPDATE)
* Activate all of the above changes
SETR RACLIST (RDATALIB) REFRESH
SETR RACLIST (DIGTRING) REFRESH
SETR RACLIST (DIGTCERT) REFRESH
SETR RACLIST (FACILITY) REFRESH
/*
```

- Controlling access by using only the FACILITY class

  Example 16 illustrates how to grant the appropriate authority to our three UIDs by using only the FACILITY class.

Example 16 Granting permission to RACF digital certificate resources by using only the FACILITY class

```
//RACDCERT EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSUADS DD DSN=SYS1.UADS,DISP=SHR
//SYSLBC DD DSN=SYS1.BRODCAST,DISP=SHR
//SYSTSIN DD *
* Activate required SAF classes and define required resources
SETROPTS CLASSACT(DIGTCERT DIGTRING)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
* Give Db2 for z/OS user IDs sufficient authority so System SSL can use RACF
cert APIs
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(DB2AUSER) ACCESS(READ)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(DB2AUSER) ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(DB2BUSER) ACCESS(READ)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(DB2BUSER) ACCESS(READ)
* Give cert management user ID sufficient access to manage certificates and key
rings
* on behalf of Db2 user IDs and for certificate authorities
PERMIT IRR.DIGTCERT.ADD CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
PERMIT IRR.DIGTCERT.ADDRING CLASS(FACILITY) ID(CERTMGR) ACC(UPDATE)
PERMIT IRR.DIGTCERT.CONNECT CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
PERMIT IRR.DIGTCERT.EXPORT CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
PERMIT IRR.DIGTCERT.MAP CLASS(FACILITY) ID(CERTMGR) ACC(UPDATE)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(CERTMGR) ACC(CONTROL)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CERTMGR) ACC(UPDATE)
* Activate all of the above changes
SETR RACLIST (DIGTRING) REFRESH
SETR RACLIST (DIGTCERT) REFRESH
SETR RACLIST (FACILITY) REFRESH
/*
```

Creating the keys, digital certificates, and key rings by using RACDCERT

We are now ready to generate the necessary keys and digital certificates.

Tip: All the listed **`RACDCERT`** commands must be issued from the designated certificate management UID (`CERTMGR` in our scenario). Those commands explicitly specify the intended owner of those certificates and key rings.

In this scenario, we use a local CA, which is the entity that is trusted by Db2 and the Db2 clients to ensure the identity of communication partners when new secure DRDA connections are established. We use z/OS RACF to serve as that CA.

To implement this scenario, we must generate the following public/private key pairs and X.509 certificates:

1. A CA key pair and certificate. The CA private key is used to sign the Db2 for z/OS subsystems' certificates, and the CA certificate contains the CA's public key so that the Db2 clients can verify the Db2 for z/OS subsystems' certificates.
2. Next, we generate a public/private key pair and certificate to identify each Db2 for z/OS subsystem to its clients. Under the direction of AT-TLS, System SSL sends this certificate (considered the server certificate) to the Db2 client during the TLS handshake. This certificate is signed by using the CA's private key, and includes the Db2 for z/OS subsystem's public key.

   All the key pairs will use 2048 RSA keys and the certificates will be signed using RSA with SHA-256 signatures. For more information on key sizes and associated signature hash sizes, see the RACDCERT GENCERT (Generate certificate) command reference.

   System SSL uses the Db2 private key to sign TLS handshake messages that are sent by the client to Db2 for z/OS. The Db2 clients use the server's public key (obtained from server certificate) to verify those signatures.

Figure 4 illustrates the required private key and certificate configuration for this scenario, assuming that TLS/SSL server authentication is used. The red arrow illustrates that the server sends its personal certificate to the client during the TLS/SSL handshake. For DB2B, we would replicate this picture by using DB2B's own key ring, private key, and certificate, but with the same CA certificate because the same CA signed all the Db2 for z/OS personal certificates.

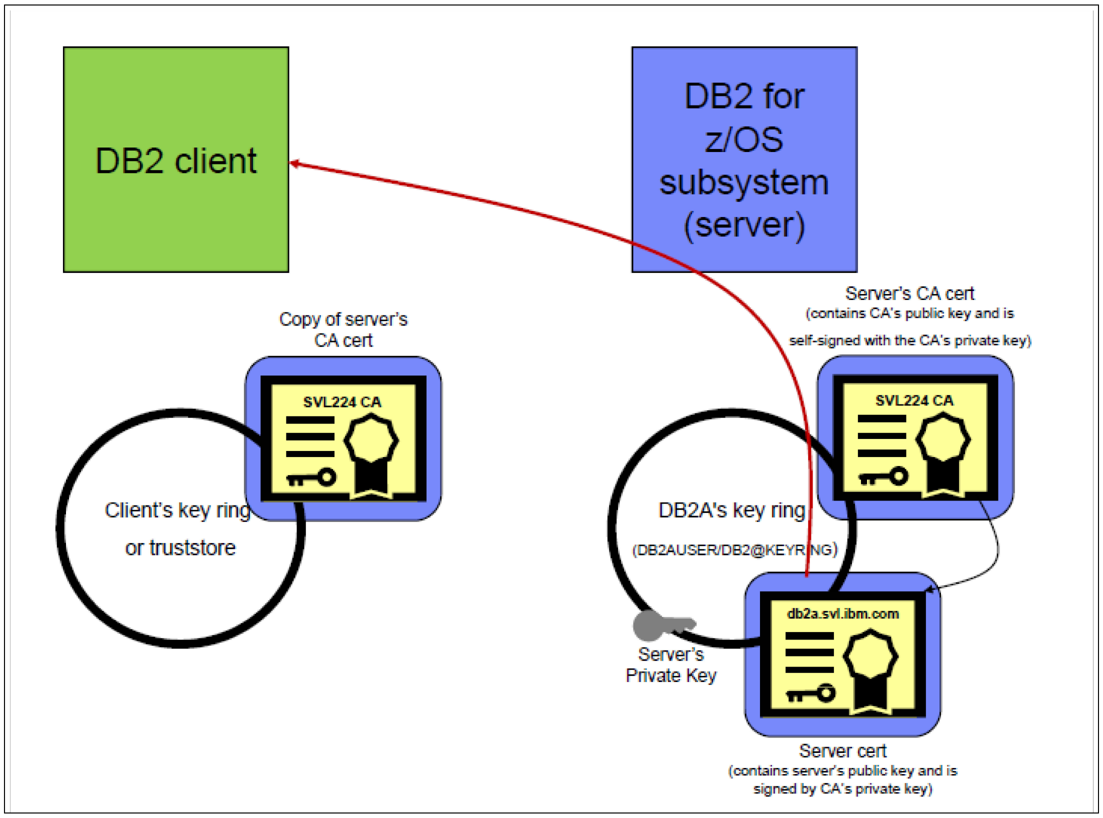

DB2 client
DB2 for z/OS subsystem (server)
Copy of server's CA cert
SVL224 CA
Client's key ring or truststore
Server's CA cert
(contains CA's public key and is self-signed with the CA's private key)
SVL224 CA
DB2A's key ring
(DB2AUSER/DB2@KEYRING)
db2a.svl.ibm.com
Server's Private Key
Server cert
(contains server's public key and is signed by CA's private key)

Figure 4 Private key and certificate configuration for CA-signed server certificate

To generate a CA key pair and certificate in RACF, use the **RACDCERT** command that is shown in Example 17. The CA certificates are not associated with any UID, so the **CERTAUTH** keyword is specified instead of **ID(userid)**.

Example 17 Generating a CA key pair and certificate

```
RACDCERT CERTAUTH -
        GENCERT  -
        RSA SIZE(2048) -
        SUBJECTSDN(OU('SVL224 CA') -
                  O('IBM') -
                  L('SVL') -
                  SP('CA') -
                  C('US')) -
        NOTAFTER(DATE(2031-12-31)) -
        WITHLABEL('SVL224 CA Cert') -
        KEYUSAGE(CERTSIGN)
```

To generate a key pair and certificate for each Db2 for z/OS subsystem (called *personal certificates* in RACF terms) that are signed by the CA private key that we created, use the **RACDCERT** command that is shown in Example 18. Each key pair and certificate is owned by the UIDs of the Db2 subsystems, allowing the Db2 subsystem and AT-TLS and System SSL acting on the subsystem's behalf to access the keys and certificate.

To support multiple Db2 subsystems with a streamlined AT-TLS policy, the same key ring name is used by each Db2 for z/OS subsystem, and each one is owned by the UID that is associated with the DB2xDIST address space. The RACF label and common name (CN) for the certificate can include the Db2 for z/OS subsystem identifiers to more clearly link each certificate to a specific Db2 subsystem. We also set a reasonable certificate expiration date, which must be tracked carefully by the certificate administrator so that the certificate can be renewed before the certificate expires.

Example 18 Generating key pairs and personal certificates for the Db2 server for z/OS subsystems

```
RACDCERT ID(DB2AUSER) -
        GENCERT -
        RSA SIZE(2048) -
        SUBJECTSDN(CN('db2a.svl.ibm.com') -
                  OU('UTEC224') -
                  O('SVL224') -
                  C('US')) -
        NOTAFTER(DATE(2022-12-31)) -
        WITHLABEL('DB2A Cert') -
        SIGNWITH(CERTAUTH LABEL('SVL224 CA Cert'))
RACDCERT ID(DB2BUSER) -
        GENCERT -
        RSA SIZE(2048) -
        SUBJECTSDN(CN('db2b.svl.ibm.com') -
                  OU('UTEC224') -
                  O('SVL224') -
                  C('US')) -
        NOTAFTER(DATE(2022-12-31)) -
        WITHLABEL('DB2B Cert') -
        SIGNWITH(CERTAUTH LABEL('SVL224 CA Cert'))
```

Now that we have generated the CA and Db2 subsystem keys and certificates, we must create key rings to hold the digital certificates and the subsystem private keys where the DB2ADIST and DB2BDIST started tasks can access them. The name of the key rings must match the value of the **`Keyring`** parameter that was specified in the **`TTLSKeyRingParms`** statement within AT-TLS policy.

Additionally, each key ring must belong to the UID under which the respective DB2xDIST address space will run. Finally, we identify each Db2 server certificate as the default certificate for its respective key ring so it is selected by default during TLS handshakes.

Example 19 lists the **`RACDCERT`** commands to create the `DB2@KEYRING` key rings and connect the required digital certificates and private keys.

Example 19 Creating the key rings and connecting the digital certificates and private keys

```
RACDCERT ID(DB2AUSER) ADDRING(DB2@KEYRING)
RACDCERT ID(DB2AUSER) CONNECT(CERTAUTH LABEL('SVL224 CA Cert') -
                              RING(DB2@KEYRING))
RACDCERT ID(DB2AUSER) CONNECT(ID(DB2AUSER) -
                              LABEL('DB2A Cert') -
                              RING(DB2@KEYRING) -
                              DEFAULT)
RACDCERT ID(DB2BUSER) ADDRING(DB2@KEYRING)
RACDCERT ID(DB2BUSER) CONNECT(CERTAUTH LABEL('SVL224 CA Cert') -
                              RING(DB2@KEYRING))
RACDCERT ID(DB2BUSER) CONNECT(ID(DB2BUSER) -
                              LABEL('DB2B Cert') -
                              RING(DB2@KEYRING) -
                              DEFAULT)
```

Important: The key ring name and the label names are case-sensitive.

Now that the keys and certificates are created and available to the DB2xDIST started tasks, the final step is to export the CA certificate (not the server certificates) into an MVS data set so that it can be shared with the appropriate Db2 clients. Each client must add or import a copy of this CA certificate to their key ring or keystore or truststore so it can be used to verify the Db2 server certificate during the TLS/SSL handshake.

RACF supports various export formats, including Distinguished Encoding Rules (DER), DER encoding with additional Base 64 encoding, Public Key Cryptography Standards (PKCS) #12, and PKCS#7. Example 20 shows the **`RACDCERT`** command to export the CA certificate by using the default format of Base64-encoded DER.

Important: The export format that you use determines whether the exported certificate must be transported to the client as binary or text data. Formats that use Base64 encoding should be transported as text data, and formats that are DER encoded should be transported as binary data. The default Base64-encoded DER takes the binary DER encoded data and then encodes it by using Base64, so it should be transported as text data.

Example 20 Exporting the CA certificate

```
RACDCERT CERTAUTH -
         EXPORT(LABEL('SVL224 CA Cert')) -
         DSN('USRT001.SVL224.CACERT')
```

Now, the AT-TLS and certificate configuration that is required for TLS/SSL server authentication is complete. If you plan to use client authentication, then continue reading. If not, then you can skip to “Configuring a secure port for the Db2 for z/OS server” on page 43.

Registering a client certificate with RACF (optional)

If TLS/SSL client authentication is configured (`HandshakeRole` is `ServerWithClientAuth`), the server must have a certificate on its key ring that can be used to verify the signature of the client certificate during the TLS/SSL handshake.

If the client’s certificate is signed by a different CA than the one that signed the server certificate, you must import that CA’s certificate into RACF and then add it to the server’s key ring. If the same CA is used to sign both the server and client certificate, no additional configuration is required because the server already has that CA certificate on its key ring.

Because you already saw an example of configuring a certificate that was signed by a CA, we use the client certificate to illustrate another widely used concept: *self-signed certificates*. For brevity's sake, we illustrate these certificates only for the DB2A server, but the same approach can be used for DB2B or any other Db2 for z/OS subsystems that you defined in your environment.

When you are first learning about certificates or when you are first trying out certificate-based authentication in a test environment, you might want to configure TLS/SSL sessions without involving a CA. In these situations, you can use self-signed certificates. A *self-signed certificate* is signed by the private key that is associated with that certificate’s public key. Essentially, the communications partner is vouching for its own identity without any third-party trust. Self-signed certificates are not recommended for production environments or test environment in which the two communicating partners do not belong to the same business entity. However, due to their usefulness in testing, the illustrated configuration shows the Db2 for z/OS subsystem establishing secure DRDA sessions with a client that uses a self-signed certificate.

Figure 5 on page 41 illustrates our previous certificate configuration with a self-signed client certificate added.

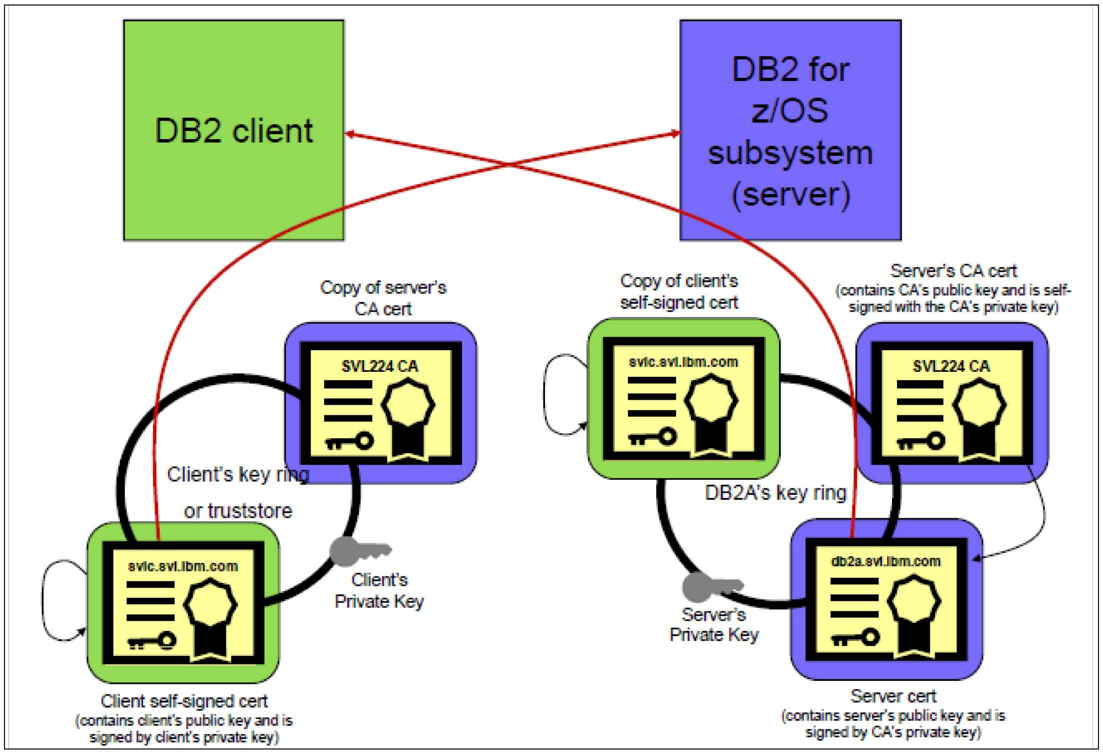

DB2 client
DB2 for z/OS subsystem (server)
Copy of server's CA cert
SVL224 CA
Client's key ring or truststore
svlc.svl.ibm.com
Client's Private Key
Client self-signed cert (contains client's public key and is signed by client's private key)
Copy of client's self-signed cert
svlc.svl.ibm.com
Server's CA cert (contains CA's public key and is self-signed with the CA's private key)
SVL224 CA
DB2A's key ring
db2a.svl.ibm.com
Server's Private Key
Server cert (contains server's public key and is signed by CA's private key)

Figure 5 Adding a self-signed client certificate to our configuration

During the TLS handshake and after the client verified the server's certificate, the client sends its own certificate to the Db2 server to identify itself. Usually, the server would use the client's CA certificate to validate the client certificate, which ensures the client's identity.

Because the client signed its certificate with its own private key in this case, the client's personal certificate serves as the signing certificate. Therefore, a copy of the client certificate is added to the server's key ring as a trusted CERTAUTH certificate before any TLS handshakes are attempted.

Example 21 illustrates the **`RACDCERT ADD`** command that adds a copy of the client's self-signed certificate to the RACF database. Assuming that the client certificate was uploaded to z/OS into a data set that is named `USRT002.CLIENT.SSCRT`, this command reads the certificate from that data set and adds it to the RACF database. The certificate in the database is classified as a CA certificate and marked as *trusted* so that System SSL trusts it to verify the client certificate that is sent during the TLS/SSL handshake.

Example 21 Adding a copy of the self-signed client certificate to RACF as a trusted CA certificate

```
RACDCERT CERTAUTH ADD('USRT002.CLIENT.SSCRT') TRUST WITHLABEL('SVLC SSCert')
```

Next, we must add this self-signed client certificate to the appropriate server key rings. In this case, we assume that the client connects only to DB2A, so we must connect the client certificate to `DB2AUSER/DB2@KEYRING`. Example 22 shows you how to do this task by using the **`RACDCERT CONNECT`** command. This command connects the client certificate in the RACF database to the server key ring `DB2@KEYRING`, which is owned by z/OS UID `DB2AUSER` under the label `‘SVLC SSCert’`.

Example 22 Connecting a self-signed client certificate to a key ring

```
RACDCERT ID(DB2AUSER) CONNECT(CERTAUTH LABEL('SVLC SSCert') -
RING(DB2@KEYRING))
```

Because the certificate is being connected as a CA certificate, we do not specify an owning UID for the certificate. Instead, we specify `CERTAUTH`, which also implies the certificate’s usage (therefore, we do not have to specify `USAGE(CERTAUTH)`).

Now, you have completed the necessary configuration steps to create the key pairs, digital certificates, and the key ring to hold the digital certificates in RACF for the TLS/SSL client authentication scenario.

Creating and activating client certificate name filters

A RACF *certificate name filter* enables you to associate one or more client certificates with one z/OS UID that is based on the unique user information in the certificate, such as the organization to which the user belongs. You can create one or more certificate name filters to map a many client certificates to a limited number of UIDs.

Certificate name filters are flexible in the way that certificate identities can be mapped to z/OS UIDs. For more information, see the description of the **`RACDCERT MAP`** command in *z/OS V2R4 Security Server RACF Command Language Reference*, SA23-2292.

Another way to associate a z/OS UID with a client certificate is by storing the client certificate in the RACF database and designating the associated z/OS UID as the certificate owner. Using certificate name filters eliminates the need to store each client certificate in the RACF database, which can reduce administrative costs.

AT-TLS **`ClientAuthType SAFCheck`** and Db2 client certificate authentication processing (as described under “Peer authentication models” on page 16) use the z/OS UIDs mapped through certificate name filters.

To create and activate a certificate name filter, complete the following steps:

1. Create a certificate name filter by issuing the **`RACDCERT MAP`** command, as shown in Example 23.

   Example 23 RACDCERT MAP command to create a certificate name filter

   ```
   RACDCERT MAP ID(USRT001) -
   SDNFILTER('O=IBM.L=San Jose.SP=CA.C=US') -
   WITHLABEL('IBMers') TRUST
   ```

   This command creates a new certificate name filter that is based on the subject distinguished name (SDN) in the certificate. The filter associates the `USRT001` UID to any user presenting a certificate with an SDN that contains `‘O=IBM.L=San Jose. SP=CA.C=US’`.

2. Activate the **SETROPTS RACLIST** processing for the DIGTNMAP class. Using the **RACDCERT MAP** command to create a certificate name filter automatically generates a mapping profile in the DIGTNMAP class that represents the new filter. Both the DIGTNMAP class and the **SETROPTS RACLIST** processing for the DIGTNMAP class must be active before the new certificate name filter can be considered complete.

   To activate the **SETROPTS RACLIST** processing for the DIGTNMAP class, run the following command:

   ```
   SETROPTS CLASSACT(DIGTNMAP) RACLIST(DIGTNMAP)
   ```

3. Refresh the DIGTNMAP class.

   When **SETROPTS RACLIST** processing for the DIGTNMAP class is active, you must refresh the DIGTNMAP class for the certificate name filter to take effect. To refresh the DIGTNMAP class, run the following command:

   ```
   SETROPTS RACLIST(DIGTNMAP) REFRESH
   ```

With these steps complete, all certificates with an SDN containing `‘O=IBM.L=San Jose.SP=CA.C=US’` are mapped to the `USRT001` z/OS UID (in these examples, it is the UID that AT-TLS and Db2 find when **`ClientAuthType SAFCheck`** is specified).

Configuring a secure port for the Db2 for z/OS server

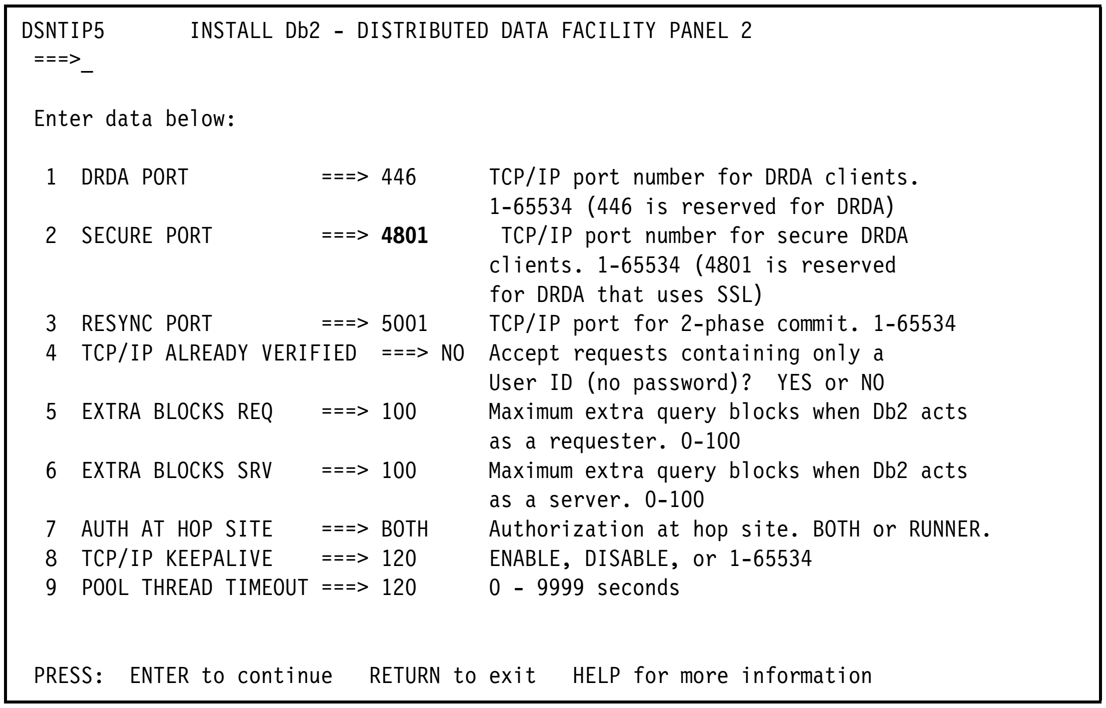
With Db2 for z/OS, DDF can (optionally) listen to a secondary secure port for inbound TLS/SSL connections. The DDF SQL TCP/IP Listener accepts unprotected (clear-text) connections on the DRDA port, but the secure port accepts only TLS/SSL connections to provide secure communications with a partner that uses TLS/SSL. Therefore, Db2 cannot accept a nonsecure TCP/IP connection through the secure port.

If an improper use of a secure port is detected, Db2 rejects the connection with a DSNL030I message with reason code 00D31205. Therefore, client connections are ensured of getting the TLS/SSL connection that they require when they connect to a Db2 for z/OS server through a secure port. The system administrator specifies the secure port number. Using the **-DISPLAY LOCATION** Db2 command shows which locations are using TLS/SSL connections.

To define a secure port to Db2, you can specify the secure port number during the Db2 installation by using the DISTRIBUTED DATA FACILITY PANEL 2 (DSNTIP5), or you can specify the secure port by updating the DDF communication record in the bootstrap data set (BSDS) by using the Change Log Inventory (**DSNJU003**) utility.

Specifying the secure port during Db2 installation

To specify the secure port during Db2 installation, specify the TCP/IP port number in the DRDA SECURE PORT field of the DISTRIBUTED DATA FACILITY PANEL 2 (DSNTIP5). For example, if you want to configure the secure port number as 4801, you specify the value `4801` in the SECURE PORT field, as shown in Figure 6.



```
DSNTIP5       INSTALL Db2 - DISTRIBUTED DATA FACILITY PANEL 2
 ===>_

 Enter data below:

  1  DRDA PORT           ===> 446      TCP/IP port number for DRDA clients.
                                       1-65534 (446 is reserved for DRDA)
  2  SECURE PORT         ===> 4801      TCP/IP port number for secure DRDA
                                       clients. 1-65534 (4801 is reserved
                                       for DRDA that uses SSL)
  3  RESYNC PORT         ===> 5001     TCP/IP port for 2-phase commit. 1-65534
  4  TCP/IP ALREADY VERIFIED  ===> NO  Accept requests containing only a
                                       User ID (no password)?  YES or NO
  5  EXTRA BLOCKS REQ    ===> 100      Maximum extra query blocks when Db2 acts
                                       as a requester. 0-100
  6  EXTRA BLOCKS SRV    ===> 100      Maximum extra query blocks when Db2 acts
                                       as a server. 0-100
  7  AUTH AT HOP SITE    ===> BOTH     Authorization at hop site. BOTH or RUNNER.
  8  TCP/IP KEEPALIVE    ===> 120      ENABLE, DISABLE, or 1-65534
  9  POOL THREAD TIMEOUT ===> 120      0 - 9999 seconds


 PRESS:  ENTER to continue   RETURN to exit   HELP for more information
```

Figure 6 DSNTIP5: INSTALL Db2 - DISTRIBUTED DATA FACILITY PANEL 2

If the DRDA SECURE PORT field is left blank, TLS/SSL verification support is disabled, and the DDF TCP/IP SQL Listener does not accept any inbound TLS/SSL connections on the secure port.

DRDA port and AT-TLS: If the secure port is disabled (that is, `SECPORT=0`), you can configure AT-TLS to protect the DRDA PORT and clients can connect to it by using TLS/SSL. However, if you do this task, Db2 is unaware of the protection and does not validate whether the connection uses TLS/SSL.

Specifying the secure port by updating the DDF communication record in the BSDS

To specify the secure port by updating the DDF communication record in the BSDS, use the **`SECPORT`** parameter in the **`DDF`** statement that is used with the Change Log Inventory (**`DSNJU003`**) stand-alone utility. The **`SECPORT`** parameter specifies the port number for the DDF TCP/IP SQL Listener to accept inbound TLS/SSL connections. For example, to configure the secure port with a value of `4801`, you code the following line:

```
DDF LOCATION=LOC1,SECPORT=4801
```

If the **`RESPORT`** value matches the value of either **`SECPORT`** or **`PORT`**, Db2 issues an error. If you specify a value of `0` for the **`SECPORT`** parameter, TLS/SSL verification support is disabled, and the DDF TCP/IP SQL Listener does not accept any inbound TLS/SSL connections on the secure port.

Data sharing considerations

For a data sharing environment, each Db2 member with TLS/SSL support must specify a secure port. The secure port for each Db2 member of the group must be the same port, and the DRDA PORT for each member also needs to be the same port. If each Db2 member specifies a unique secure port, the behavior is unpredictable. For example, sysplex member workload balancing might not function correctly.

Similarly, for Db2 members that are defined as a subset of the data sharing group, each Db2 member that belongs to the subset must configure the alias secure port. Use the **`DSNJU003`** stand-alone utility and specify the value of the alias secure port in the **`ALIAS`** keyword of the **`DDF`** statement. For example, you might specify the following **`DDF`** statement to define an alias secure port of `6448` for a Db2 member that belongs to a subset:

```
DDF LOCATION=LOC1,SECPORT=4801,ALIAS=LOC1ALIAS1:6446:6448
```

Tip: If you want to define a location alias name only for a Db2 member, you do not need to define a separate unique secure port for the location alias.

Db2 for z/OS provides an alternative method to define an alias secure port by using the **`-MODIFY DDF`** command. For example, to specify the secure port dynamically for a defined alias for a Db2 member that belongs to a subset, you can issue the following Db2 command:

```
-MODIFY DDF ALIAS(LOC1ALIAS1) SECPORT(6448)
```

If the alias is not ready to accept connections, issue the following Db2 command:

```
-MODIFY DDF ALIAS(LOC1ALIAS1) START
```

In a data sharing environment, your `PROFILE.TCPIP` contains **`PORT`** statements for DB2xDIST job names. If these **`PORT`** statements also specify the **`BIND ipaddr`** option, you must remove the **`BIND ipaddr`** specification from the **`PORT`** statement to configure Db2 to accept secure connections through the secure port. The DDF SQL TCP/IP Listener listens for TLS/SSL inbound socket requests only on an IP address that is bound to `INADDR_ANY` (IPv4) or `in6addr_any` (IPv6).

If DDF detects that an IP address other than `INADDR_ANY` or `in6addr_any` is bound to the secure port, DDF prevents the DDF SQL TCP/IP Listener from accepting secure connections through the secure port. The message DSNL512I condition and the error message "`BINDSPECIFIC NOT SUPPORTED WITH SECURE PORT`" are issued on the system console.

If you must configure specific IP addresses for each Db2 member (for example, a dynamic virtual IP address (DVIPA) configuration), you can use the BSDS DDF communication record statements **`IPV4/GRPIPV4`** or **`IPV6/GRPIPV6`** to define the specific IP addresses for each member. You can use the group IP address with the IP addresses that are specified for the **`PORT`** statement from the TCP/IP profile (`PROFILE.TCPIP`). Then, you remove the specification of the IP addresses from the **`PORT`** statement in `PROFILE.TCPIP`.

Configuring Db2 for z/OS as a requester with TLS/SSL support

A Db2 for z/OS subsystem can also act as a Db2 requester. In this case, Db2 for z/OS must be able to initiate TLS-protected connections to certain servers. To ensure TLS-protected connections, you can make communications database (CDB) changes that indicate that TLS-protected connections are required to certain remote locations. If a secure connection is required, DDF must determine whether an AT-TLS policy rule is defined and AT-TLS is enabled for the outbound connection.

DDF validates the secure status of the outbound connection by querying AT-TLS. Therefore, if AT-TLS indicates that the status of the connection is not secure and a secure connection is required, DDF does not establish the outbound connection with the target server. SQLCODE -904 with reason code 00D31205 is returned to the application.

Changes in the Db2 communication database

To configure an outbound connection that requires a secure connection, insert a row in the `SYSIBM.LOCATIONS` table and specify the value of `Y` in the `SECURE` column. You also must specify a port number in the `PORT` column for the outbound connection to use. For secure connections, the value of the `PORT` column must take the value of the configured secure port at the target server. However, if the value of the `PORT` column is blank *and* the value of the `SECURE` column specifies `Y`, DDF uses the reserved secure port (4801) as the default.

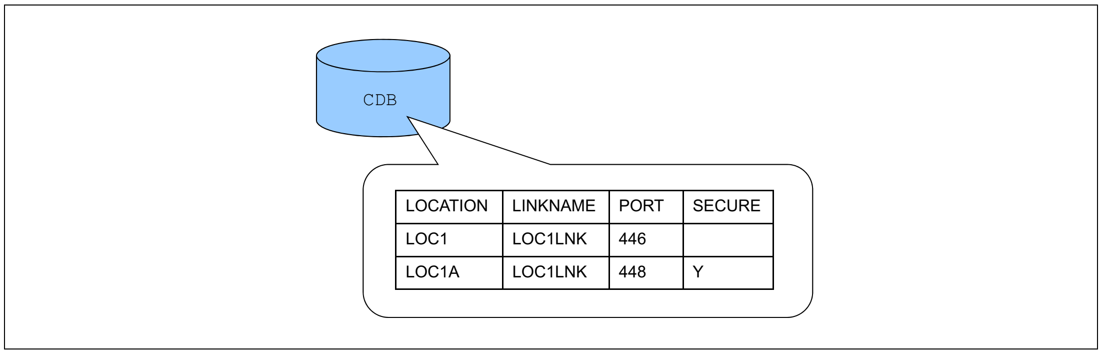
LOCATION ALIAS name considerations for TLS/SSL support

Certain Db2 applications might require TLS/SSL protection and accept the performance cost for this level of security. However, applications might be satisfied with unprotected connections. You can achieve this flexibility by using the LOCATION ALIAS name feature.

Consider a Db2 server that is configured to support both nonsecure and secure connections. At the Db2 requester, you can define two rows in the `SYSIBM.LOCATIONS` table:

- One row that specifies the location name and the nonsecure DRDA port of the server.
- One row that specifies a separate location name, the secure DRDA port of the server, and `SECURE='Y'`.

At the Db2 server, you can define a LOCATION ALIAS name to provide an alternative name for any Db2 requesters that need to access the server by using TLS/SSL protection, as shown in Figure 7.



LOCATION	LINKNAME	PORT	SECURE
LOC1	LOC1LNK	446	
LOC1A	LOC1LNK	448	Y

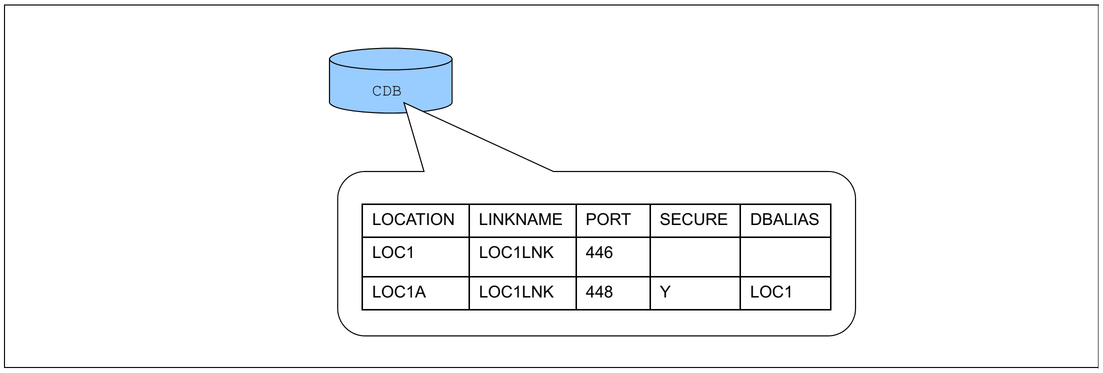
Figure 7 LOCATION ALIAS consideration

DBALIAS name considerations for TLS/SSL support

Using the LOCATION ALIAS name feature as described requires server changes. If your system environment does not enable you to perform the server changes (to define an alias name for the server), an alternative solution to using the LOCATION ALIAS name feature is to use the DBALIAS name feature.

Using the same scenario, go to the Db2 requester and define two rows in the `SYSIBM.LOCATIONS` table, as shown in Figure 8:

- A row that specifies the location name and the nonsecure DRDA port of the remote server.
- Another row that specifies a separate location name, which is the secure DRDA port `SECURE='Y'` for the remote server, and a database alias (DBALIAS) name that refers to the location name from the previous row.



LOCATION	LINKNAME	PORT	SECURE	DBALIAS
LOC1	LOC1LNK	446		
LOC1A	LOC1LNK	448	Y	LOC1

Figure 8   LOCATION DBALIAS name feature

Configuring the PAGENT and AT-TLS

At the Db2 requester, the PAGENT and AT-TLS must be configured on the requester's z/OS system. A single Db2 for z/OS subsystem can act as both a server and requester simultaneously, so we build on the sample policy from Example 11 on page 18 to illustrate how to add a requester rule for DB2A.

Defining AT-TLS policy rules

The steps to define the AT-TLS policy rules are the same as the steps that are described in "Defining the AT-TLS policy rules" on page 14 for defining AT-TLS policy rules for the Db2 server role, except for these **`TTLSRule`** conditions:

- `RemotePortRange = 4801`
- `Direction = Outbound`
- `HandshakeRole = Client`

Example 24 provides a simple AT-TLS rule that can be added to the policy in Example 11 on page 18 to allow DB2A to initiate secure outbound connections to remote servers listening on port 4801.

Example 24   AT-TLS policy statements to allow Db2 requester traffic from DB2A

```
TTLSRule DB2ASecureRequester
{
  RemotePortRange 4801
  JobName DB2ADIST
  Direction Outbound
  TTLSGroupActionRef DB2@GrpAct
  TTLSEnvironmentActionRef DB2@SecureClientEnvAct
}
TTLSEnvironmentAction DB2@SecureClientEnvAct
{
  HandshakeRole Client
  TTLSKeyRingParmsRef DB2@KeyRing
  TTLSEnvironmentAdvancedParmsRef DB2@EnvAdvParms
  TTLSCipherParmsRef DB2@CipherParms
  TTLSSignatureParmsRef DB2@SigParms
}
```

Digital certificate and key ring considerations

To complete any TLS/SSL handshake with a server, the client system must have a key ring that contains a certificate that can be used to authenticate the server's certificate, which in this example is the server's CA certificate. For illustration purposes, we assume that the remote Db2 server to which DB2A will connect uses a certificate that is signed by a different CA, and that CA's certificate was exported into a data set that is named `USRT050.RTP853.CACERT` and transferred by using FTP to the local z/OS system by using the same data set name.

Use the **`RACDCERT`** command to store the server's CA certificate in the local RACF database, and add that certificate to the DB2AUSER's key ring. Example 25 shows the **`RACDCERT`** commands to perform these tasks.

Example 25   Storing and adding the remote Db2 server's CA certificate

```
RACDCERT CERTAUTH ADD('USRT050.RTP853.CACERT') TRUST -
        WITHLABEL('RTP853 CA Cert')
RACDCERT ID(DB2AUSER) CONNECT(CERTAUTH -
                         LABEL('RTP853 CA Cert') -
                         RING(DB2@KEYRING))
```

The final step is to update the PAGENT at the client system by using the MVS **`MODIFY`** command or by using the MVS **`START`** command if the PAGENT is not started.

If the server to which DB2A will connect is configured for TLS/SSL server authentication, then our configuration is complete.

If the server is configured for TLS/SSL client authentication, it needs a copy of DB2A's CA certificate added to its key ring or truststore if it is not already there. In this scenario, DB2A, which is the TLS/SSL client, must present its own certificate to the remote server during the TLS/SSL handshake. Because DB2A's personal certificate is already present on its key ring (with label **"**`DB2A Cert`**"**), it is already configured correctly.

Configuring Java applications by using IBM DS Driver for JDBC and SQLJ to use TLS/SSL

In this section, we describe the steps to configure a Java application that uses IBM DS Driver for JDBC and SQLJ (Type 4 connectivity) to access a Db2 10 for z/OS (or later versions) server by using TLS/SSL. Two configuration steps are necessary:

- Configuring connections under the IBM DS Driver for JDBC and SQLJ to use TLS/SSL
- Configure the Java Runtime Environment (JRE) to use TLS/SSL

Keystore and truststore versus z/OS key rings

Before we get into the details of Java-based certificate configuration, we briefly must explain some terminology and concepts that are behind storing the relevant certificates and keys.

On z/OS, we stored all our digital certificates and private keys on a RACF key ring. There are two purposes that are served by these key rings:

- Storage of one's personal certificate and the private key that is associated with that certificate. We send our personal certificate to a remote peer during a TLS/SSL handshake to prove our identity to that remote peer. Our private key (which is only known to us) is used to sign certain TLS/SSL handshake messages and decrypt messages that are encrypted by remote peers by using our public key, which they find in the personal certificate that we send during the TLS/SSL handshake.
- Storage of certificates that we trust for verifying the authenticity of the remote peer's personal certificate during a TLS/SSL handshake, which can be CA certificates, self-signed peer certificates, or both.

Although a RACF key ring fulfills both of these purposes, Java's TLS/SSL implementation uses two separate repositories to do so:

- A *keystore* contains a Java application's personal certificate and private key.
- A *truststore* contains the certificates that will be used to verify the authenticity of peer certificates (CA certificates and self-signed peer certificates).

As we see, the configuration of the keystore and truststore is as critical to the Java TLS/SSL configuration as key ring configuration is to z/OS TLS/SSL configuration.

Configuring connections under the IBM DS Driver for JDBC and SQLJ to use TLS/SSL

To configure connections under the IBM DS Driver for JDBC and SQLJ to use TLS/SSL, you must set the `DB2BaseDataSource.SetSslConnection` property to `true`.

Optional: Set `DB2BaseDataSource.SetSslTrustStoreLocation` on a Connection or DataSource instance to the location of the truststore. Setting the `sslTrustStoreLocation` property is an alternative to setting the Java `javax.net.ssl.trustStore` property. If you set `DB2BaseDataSource.SetSslTrustStoreLocation`, the `javax.net.ssl.trustStore` property is not used.

The code sample in Example 26 shows how to set the `sslConnection` property on a Connection instance.

Example 26 The sslConnection property

```
java.util.Properties props = new java.util.Properties();
props.put( "user", user ID );
props.put( "password", passwd );
props.put( "sslConnection", "true" );
java.sql.Connection con =
  java.sql.DriverManager.getConnection(url, props);
```

Configuring the Java Runtime Environment to use TLS/SSL

Before you can use TLS/SSL with your JDBC and SQLJ applications, you must configure the JRE to use TLS/SSL. Consider these prerequisites:

- The JRE must include a Java security provider. You must install either the IBM Java Secure Socket Extension (JSSE) provider or the Sun JSSE provider. The IBM JSSE provider is automatically installed with the IBM Software Development Kit (SDK) for Java.

  Restriction: You cannot use the Sun JSSE provider with the IBM JRE. If you use the Sun JSSE provider, it works only with the Sun JRE.

- TLS/SSL support must be configured at the target server.

We show a configuration for server authentication and a configuration for client authentication.

Configuring the JRE to use TLS/SSL server authentication

The following steps demonstrate how to set up the JRE to use TLS/SSL server authentication:

1. Obtain the Db2 for z/OS server's CA certificate and store it in a Java truststore. One method is to use the **FTP** command to **GET** the file. Recall that because we exported the certificate in Example 20 on page 39 into Base64-encoded DER format, we must perform the `FTP GET` command in ASCII mode.

   Important: Do not use BINARY mode transmission to get the certificate if it is exported in BASE64-encoded DER format.

   Use the Java **keytool** command and specify the **-import** option to import the certificate into the truststore. If you want to create your own truststore, use the **-genkey** option with the Java **keytool** command to create your keystore before you import the certificate. For example, to create a truststore named *myTrustStore*, run the following command:

   ```
   keytool -genkeypair -keystore myTrustStore
   ```

   Creating your own truststore is an optional step. The default truststore that is included with JSSE is named `<`*java-home*`>/lib/security/jssecacerts` or `<`*java-home*`>/lib/security/cacerts`. To create your own truststore, let's assume you downloaded the server's CA certificate from the USRT001.SVL224.CACERT dataset created in Example 20 on page 39 to a file named svl224.cacert on the client system. To import the server's CA certificate into the truststore just created and assign it the alias (analogous to the RACF certificate label) svl224_ca, use the following command:

   ```
   keytool -importcert -file svl224.cacert -trustcacerts -keystore myTrustStore
   -alias svl224_ca
   ```

   If you do not specify the **-keystore** option on the **keytool** command, the server's CA certificate is stored in the default truststore.

2. To verify that the server's CA certificate was imported correctly into the truststore, you can display the contents of the truststore by using the **-list** option for the **keytool** command. For example, to display the contents of the truststore for the alias `svl224_ca`, run the following command:

   ```
   keytool -list -alias svl224_ca -keystore myTrustStore -v
   ```

3. If you defined your own truststore, you can use these Java system properties to specify the truststore for your Java application to use:

   – `javax.net.ssl.trustStore`: Specifies the name of the customized truststore that you specified with the **`-keystore`** parameter in the **`keytool`** utility.

     Important: If the IBM DS Driver for JDBC and SQLJ `IDB2BaseDataSource.SetSslTrustStoreLocation` property is set, its value overrides the `javax.net.ssl.trustStore` property value.

   – `javax.net.ssl.trustStorePassword` (optional): Specifies the password for the truststore. We advise you to specify a password for the truststore. If not, you cannot protect the integrity of the truststore.

     Important: If the IBM DS Driver for JDBC and SQLJ `DB2BaseDataSource.SetSslTrustStorePassword` property is set, its value overrides the `javax.net.ssl.trustStorePassword` property value.

   You can set these system properties either statically or dynamically:

   – To set these system properties statically, use the **`-D`** option of the **`java`** command. For example, to run an application that is named `testssl.java` and set the `javax.net.ssl.trustStore` system property to specify a truststore that is named `myTrustStore`, run the following command:

   ```
   java -Djavax.net.ssl.trustStore=myTrustStore testssl
   ```

   – To set these system properties dynamically, call the **`java.lang.System.setProperty`** method in your code:

   ```
   System.setProperty("javax.net.ss.trustStore", "myTrustStore");
   ```

Tip: The default TLS protocol is determined by the Java SE Development Kit. The JCC driver does not modify or change the default TLS protocol set by the Java SE Development Kit. You can override this Java SE Development Kit default by using either of two ways:

- Modify the `jdk.tls.disabledAlgorithms` property in the `%JAVA_HOME/jre/lib/security/java.security` file to disable/enable the required TLS protocols.
- Explicitly set the system property `https.protocols` by using the **`-D`** option from the **`java`** command.

  Example: Change the `java.security` file on your local workstation to disable the TLS algorithm:

  a. Create a local `java.security` file in your working directory.

  b. Add the following statement:

  ```
  jdk.tls.disabledAlgorithms=TLSv1.2, TLSv1.3
  ```

  The single equal sign means that this statement appends this `java.security` statement to the master `java.security` file when you run your client application. If you want to override the master `java.security` file, then you can specify double equal signs instead.

  c. Run your Java client application by specifying the **`-D`** option of the **`java`** command to specify the local `java.security` file to use. For example:

  ```
  java -Djava.security.properties=java.security myApp
  ```

An alternative to using the `jdk.tls.disabledAlgorithms` security property is to use the `jdk.tls.client.protocols` property to set the default TLS protocol for the JVM to use when running your Java client application.

Example: Set the default client TLS protocol by using the `jdk.tls.client.protocols` property by specifying the **`-D`** option of the `java` command. For example:

```
java -Djdk.tls.client.protocols=TLSv1.1 myApp
```

Important: If you do not set the TLS protocol when you use JDBC driver 4.31.10 or later and the AT-TLS policy of the z/OS server does not support TLSv1.2 or TLSv1.3, your connection will be unsuccessful, and your client application will receive `ERRORCODE -4499 SQLSTATE(08001)`. For example:

```
Unexpected
SQLException:com.ibm.db2.jcc.am.DisconnectNonTransientConnectionException:
[jcc][t4][2030][11211][4.31.10] A communication error occurred during
operations on the connection's underlying socket, socket input stream, or
socket output stream.  Error location: Reply.fill() - socketInputStream.read
(-1).  Message: The server selected protocol version TLS11 is not accepted by
client preferences [TLS13, TLS12]. ERRORCODE=-4499, SQLSTATE=08001
```

Configuring the JRE to use TLS/SSL client authentication

To configure the JRE to use TLS/SSL client authentication, complete the following steps:

1. Complete all the steps in “Configuring the JRE to use TLS/SSL server authentication” on page 50.
2. Enable IBM Data Server Driver for JDBC and SQLJ certificate authentication by specifying `DB2BaseDataSource.TLS_CLIENT_CERTIFICATE_SECURITY` as the value of the `securityMechanism Connection` or `DataSource` property. There are three options:
   a. Setting the `DataSource` property:
      ```
      ds.setSecurityMechanism((com.ibm.db2.jcc.DB2BaseDataSource.TLS_CLIENT_CERTIF
      ICATE_SECURITY));
      ```
   b. Setting the global properties file: `db2.jcc.securityMechanism=18`
   c. Setting the connection string: `securityMechanism=18;`
3. Use the Java **`keytool`** command to generate a self-signed client certificate. For example, the following **`keytool`** command generates an RSA algorithm key pair and self-signed certificate in the keystore that is named `myKeyStore` by using an alias of `clnt_cert`:

   ```
   keytool -genkeypair -alias clnt_cert -keyalg rsa -keystore myKeyStore
   -storepass aSecurePassWord
   ```

   This example uses the **`-keyalg`** RSA, which defaults to a key size of 2048 bits and a sigalg of SHA256withRSA.

   The keystore is password-protected, which is important because the private key is stored in this file.

   Follow the prompts to complete the generation of the client certificate.
4. Export the client certificate by using the Java **`keytool`** command. For example, use this command:

   ```
   keytool -exportcert -rfc -alias clnt_cert -file client.ca -keystore myKeyStore
   -storepass aSecurePassWord
   ```

   The output from this command produces a file that is called `client.ca`, which contains the client certificate in Base64-encoded format (the **`-rfc`** parameter causes this encoding format).
5. You might want to verify the client certificate by using the **`keytool -printcert`** Java command. For example, to verify the exported `client.ca` file, issue the following Java **`keytool`** command:

   ```
   keytool -printcert -v -file client.ca
   ```

   Figure 9 shows a sample of the output from this command.

```
C:\Dev\JCC>keytool -printcert -v -file client.ca
Owner: CN=Bob, OU=DB2, O=IM, L=San Jose, ST=CA, C=US
Issuer: CN=Bob, OU=DB2, O=IM, L=San Jose, ST=CA, C=US
Serial number: 4dfc050d
Valid from: Fri Jun 17 18:53:17 PDT 2021 until: Thu Sep 15 18:53:17 PDT 2021
Certificate fingerprints:
         MD5:  C5:84:FA:0E:CD:CB:17:9B:B4:1F:0F:4E:90:AA:0C:F4
         SHA1: EC:4F:88:BC:FA:A8:C7:17:4F:B6:9C:5B:87:6C:8D:2B:8D:9A:CA:2C
         Signature algorithm name: SHA1withRSA
         Version: 3
```

Figure 9 Sample output from the keytool -printcert command

6. Transfer the exported `client.ca` client certificate file to the z/OS system and register it with RACF. One method is to use the **`FTP`** command to **`PUT`** the file (`client.ca`) in ASCII mode.

   Important: Do not use BINARY mode transmission to **`PUT`** the certificate.

   To register the client certificate with RACF, see "Registering a client certificate with RACF (optional)" on page 40.

7. Use these Java system properties to specify the keystore for your Java application to use:

   – `javax.net.ssl.Keystore`: Specifies the name of the customized keystore that you specified with the **`-keystore`** parameter in the **`keytool`** utility.

     Important: If the IBM DS Driver for JDBC and SQLJ `DB2BaseDataSource.SetSslKeyStoreLocation` property is set, its value overrides the `javax.net.ssl.Keystore` property value.

   – `javax.net.ssl.keyStorePassword` (optional): Specifies the password for the keystore. Specify a password for the keystore. If not, you cannot protect the integrity of the keystore or your private key.

     Important: If the IBM DS Driver for JDBC and SQLJ `DB2BaseDataSource.SetSslKeyStorePassword` property is set, its value overrides the `javax.net.ssl.keyStorePassword` property value.

   You can set these system properties either statically or dynamically:

   – To set the system property statically, use the **`-D`** option of the **`java`** command. For example, to run an application that is named `testssl.java` and set the system property `javax.net.ssl.Keystore` for the named `myKeyStore`, issue the following **`java`** command:

     ```
     java -Djavax.net.ssl.Keystore=myKeyStore testssl
     ```

   – To set the system property dynamically, call the `java.lang.System.setProperty` method in your program:

     ```
     System.setProperty("javax.net.ssl.Keystore", "myKeyStore");
     ```

Configuring the IBM DS Driver non-Java interfaces: Command-line interface, ODBC, and .NET

Starting with Db2 9.5 Fix Pack 2 or later, you can configure non- Java Db2 clients for Linux, UNIX, and Microsoft Windows, such as .NET Data Provider, command-line processor (CLP), and command-line interface (CLI) to use TLS/SSL for communications with a Db2 10 for z/OS (or later) server. Both Java and non- Java interfaces are delivered together by the same product. We describe the following configuration steps in the next sections:

- Ensuring that the IBM Global Security Kit is available
- Obtaining the server's CA certificate and adding it to your client key database
- Configuring the appropriate client configuration parameter to use TLS/SSL

With Db2 10.5 Fix Pack 5 or later, the IBM Global Security Kit runtime libraries (not the CLI executable files) are included and installed with the driver installation. Therefore, you do not need to download and install the IBM Global Security Kit separately. Furthermore, you do not need to create explicitly the keystore database and add the server certificate into a keystore database manually because the driver takes care of this task for you automatically by creating a keystore database internally and adding the server certificate that is provided through the **SSLServerCertificate** keyword to the created keystore database.

For Db2 11.5.6.0 or later, the embedded SQL driver and CLP work by using this method. For more information, see the following IBM Documentation:

- https://www.ibm.com/docs/en/db2/11.5?topic=keywords-securitytransportmode
- https://www.ibm.com/docs/en/db2/11.5?topic=keywords-sslservercertificate

Configuring the IBM DS Driver non- Java interface to use TLS/SSL client authentication

If you want to use SSL client authentication, you must ensure that the IBM Global Security Kit is downloaded and installed in your client environment because you manually must create the keystore database and add the server CA certificate and client certificate with a private key into the keystore database by using the **GSKCapicmd** tool. Then, you must specify and configure the following two keywords along with the **keydb name** and **keydb stash** keywords:

- `authentication=certificate` (for more information, see https://www.ibm.com/docs/en/db2/11.5?topic=keywords-authentication.)
- `SSLClientLabel=<label for the client certificate>` (for more information, see https://www.ibm.com/docs/en/db2/11.5?topic=keywords-sslclientlabel.)

Ensuring that the IBM Global Security Kit is available

For non- Java Db2 clients for the Linux, UNIX, and Windows environments, you use the IBM GSKit to administer digital certificates and key databases. The IBM GSKit is installed automatically on the computer system during the installation of the IBM Db2 Connect server. If you do not have the IBM GSKit installed, you can install the IBM GSKit libraries from the *IBM Db2 Support Files for SSL Functionality* DVD. Alternatively, you can install the IBM GSKit libraries from an image that you download from IBM Passport Advantage®.

You also must ensure that these environment settings are configured, depending on the client platform:

- On a Windows (32-bit or 64-bit) platform, verify that the path to the IBM GSKit libraries is included in the **PATH** environment variable. For example, on Windows, add the IBM GSKit `bin` and `lib` directories to the **PATH** environment variable:

  ```
  set PATH="C:\Program Files\IBM\gsk8\bin";"C:\Program Files\IBM\gsk8\lib";%PATH%
  ```

- On Linux and UNIX platforms, the IBM GSKit libraries are in `sqllib/lib` or `sqllib/lib64`. To set the **LIBPATH**, **SHLIB_PATH**, or **LD_LIBRARY_PATH** environment variables, you run the following commands:

  ```
  export LIBPATH=$INSTHOME/sqllib/lib:$LIBPATH:.
  export SHLIB_PATH=$INSTHOME/sqllib/lib:$SHLIB_PATH:.
  export LD_LIBRARY_PATH=$INSTHOME/sqllib/lib:$LD_LIBRARY_PATH:.
  ```

Alternatively, if you want to use the IBM Key Management tool iKeyman, you must set the **JAVA_HOME** environment variable before you start iKeyman. For information about how to navigate the IBM Key Management tool to create a client key database and add the server's CA certificate to the client key database, see the following resource:

https://www.ibm.com/docs/en/SSYKE2_8.0.0/com.ibm.java.security.component.80.doc/security-component/iKeyman.8.User.Guide.pdf.

The keys, certificates, and key database always must conform to FIPS 140 mode (which is specified by the **-fips** parameter on the **gsk8capicmd** command) when they are used for TLS/SSL communication.

IBM GSKit key database versus z/OS key rings

On z/OS, we v stored all our digital certificates and private keys on a RACF key ring. There are two purposes that are served by these key rings:

- Storage of one's personal certificate and the private key that is associated with that certificate. Recall that we send our personal certificate to a remote peer during a TLS/SSL handshake to prove our identity to that remote peer. Our private key (which is known only to us) is used to sign certain TLS/SSL handshake messages and to decrypt messages that are encrypted by remote peers by using our public key (which they find in the personal certificate that we send during the TLS/SSL handshake).
- Storage of certificates that we trust that are used to verify the authenticity of the remote peer's personal certificate during a TLS/SSL handshake (CA certificates and self-signed peer certificates).

Like RACF, the IBM GSKit key fulfills both of these purposes by using a single repository that is called a *key database*. Configuration of the IBM GSKit key database is as critical to Db2 TLS/SSL configuration on Linux, UNIX, and Windows as key ring configuration is to z/OS TLS/SSL configuration.

Obtaining the server's CA certificate and adding it to your client key database

To prepare a Db2 client for Linux, UNIX, and Windows to access Db2 for z/OS by using TLS/SSL, complete the following steps:

1. Obtain the server's CA certificate from the target Db2 server. One method is to use the **FTP** command to **GET** the file. Recall that because we exported the certificate in Example 20 on page 39 into Base64-encoded DER format, we must run the **FTP GET** command in ASCII mode.

   Important: Do not use BINARY mode transmission to get the certificate if it is exported in Base64-encoded DER format.

2. On the Db2 client computer, use the **gsk8capicmd** tool to create a client key database of the CMS type. The **gsk8capicmd** tool is a non- Java -based CLI tool. Therefore, it does not require Java to be installed on your computer to run it.

   For example, to create a client key database named "mydbclnt.kdb" with a password of "myClntPwOrd" of type CMS in the directory $DB2PATH\security\keystore, run the following command:

   ```
   gsk8capicmd -keydb -create -db "mydbclnt.kdb" -pw "myClntPwOrd" -type CMS
   -stash -fips
   ```

Specifying the **-stash** option creates a stash file for storing the password to the key database after creation. A *stash file* is an automatic way of providing the password to access the key database. Also, the password is encrypted when stored in the stash file so that when access to the key database is required, the system decrypts the password from the stash file and uses it as input to access the key database. The stash file name has the following format:

```
<key database>.sth
```

You must specify the location of the stash file when accessing the key database.

3. To add the server's CA certificate to the key database that was created in step 2 on page 56, run the **gsk8capicmd** command.

   For example, to add the server's CA certificate from the `ec754.cacert` rile into the key database that is named `mydbclnt.kdb`, run the following command:

   ```
   gsk8capicmd -cert -add -db "mydbclnt.kdb" -pw "myClntPw0rd" -file
   "ec754.cacert" -label "EC754 CA" -format ascii
   ```

   To verify that the server's CA certificate was added into the key database successfully, run the following command:

   ```
   gsk8capicmd -cert -details -db "mydbclnt.kdb" -pw "myClntPw0rd" -label "EC754 CA"
   ```

Configuring the appropriate client configuration parameter to use TLS/SSL

Complete the following steps:

1. Set the appropriate connection strings or configuration parameters for your client application:

   a. CLP clients:

      i. For CLP clients, you run the **CATALOG TCPIP NODE** statement with the **SECURITY** keyword set to `SSL` to specify SSL for the connection, as shown in the following example:

      ```
      db2 catalog tcpip node tcp754as remote fvtec754.svldev.svl.ibm.com server
      4801 security ssl

      db2 catalog db db754as as stl754as at node tcp754as authentication server

      db2 catalog dcs db db754as as stlec1
      ```

      Important: The port value for the **SERVER** keyword must identify the target server's secure port. If not, the connection fails and error SQL30081N is returned.

      ii. Define the **ssl_clnt_keydb** configuration parameter with the complete path name of the client key database file (`.kdb`) and the **ssl_clnt_stash** configuration parameter with the fully qualified path name of the client key database stash file (`.sth`).

      iii. Update the Database Manager Configuration, as shown in the following example:

      ```
      db2 update dbm cfg using SSL_CLNT_KEYDB "c:\program
      files\ibm\sqllib\security\keystore\mydbclnt.kdb" SSL_CLNT_STASH
      "c:\program files\ibm\sqllib\security\keystore\mydbclnt.sth"
      ```

      iv. Connect to the server by using TLS/SSL from the CLP client:

      ```
      db2 connect to stl754as user <userid> using <password>
      ```

b. Alternatively, an embedded SQL client can use the following statement to connect to the server:

```
strcpy(dbAlias,"stl754as");
  EXEC SQL CONNECT TO :dbAlias USER :user ID USING :pwd
```

c. CLI/ODBC applications:

   i. Depending on which environment your CLI application runs, you can use either the connection string parameters (**`ssl_client_keystoredb`** and **`ssl_client_keystash`**) or Db2 configuration parameters (**`ssl_clnt_keydb`** and **`ssl_clnt_stash`**) to specify the fully qualified path to the client key database and the stash file.

      For example, if your CLI application uses the IBM Data Server Driver for ODBC and CLI, use the connection string parameters to specify the client key database and stash file.

   ii. Call **`SQLDriverConnect`** with a connection string that contains the **`SECURITY=SSL`** keyword:

```
"Database=stl754as; Protocol=tcpip; Hostname=fvtec754.svldev.svl.ibm.com;
Servicename=4801; Security=ssl; Ssl_client_keystoredb=c:\program
files\ibm\sqllib\security\keystore\mydbclnt.kdb;
Ssl_client_keystash=c:\program
files\ibm\sqllib\security\keystore\mydbclnt.sth;"
```

      For example, if your CLI application uses the IBM Data Server Client or IBM Data Server Runtime Client, you can use either the connection string parameters or Db2 configuration parameters to specify the fully qualified path to the client key database and stash file.

      To set the connection string parameter in the `db2cli.ini` file, use the following settings:

```
[stl754as]
Database=stl754as
Protocol=tcpip
Hostname=fvtec754.svldev.svl.ibm.com
Servicename=4801
Security=ssl
SSL_client_keystoredb=c:\program
files\ibm\sqllib\security\keystore\mydbclnt.kdb
SSL_client_keystash= c:\program
files\ibm\sqllib\security\keystore\mydbclnt.sth
```

      Parameters: If the connection string parameters are used, these values override the values of the Db2 configuration parameters.

d. Db2 Data Provider for .NET Framework applications

   A Db2 Data Provider for .NET Framework application can establish a TLS/SSL connection to a remote server by specifying the **`SECURITY`** connection string keyword with the value `SSL` to use TLS/SSL for the connection to the remote server. With Db2 9.7 and later, you also can specify the fully qualified path for the client key database file (`.kdb`) and for the stash file (`.sth`) by using the **`SSLClientKeystoredb`** and **`SSLClientKeystash`** connection string parameters.

Configuring remote client applications to use TLS/SSL through a Db2 Connect server for Linux, UNIX, and Windows

In this section, we describe the configuration steps to set up remote client applications (Java applications or non- Java Db2 applications) to use TLS/SSL to access a Db2 for z/OS server through a Db2 Connect server for Linux, UNIX, and Windows.

In certain client environments, remote client applications that must access a Db2 for z/OS server system can do so only through a Db2 Connect server. Furthermore, the remote client applications might be required to use TLS/SSL to connect to the Db2 Connect server because data transmission can occur over an open network.

However, if the Db2 Connect server and the Db2 for z/OS server communicate with each other over a closed network (that is, behind a secure firewall), the connections between the Db2 Connect server and the Db2 for z/OS server do not necessarily require TLS/SSL. The network is protected by a firewall or other sufficient network security technologies. As a result, in this type of environment, we must perform the following configuration steps, which are described in the next sections:

1. IBM GSKit key database versus z/OS key rings
2. Configuring TLS/SSL support in a Db2 Connect server for Linux, UNIX, and Windows
3. Configuring Java or non- Java applications to use TLS/SSL to access the Db2 Connect server

IBM GSKit key database versus z/OS key rings

On z/OS, we stored all our digital certificates and private keys on a RACF key ring. There are two purposes that are served by these key rings:

- Storage of one's personal certificate and the private key that is associated with that certificate. Recall that we send our personal certificate to a remote peer during a TLS/SSL handshake to prove our identity to that remote peer. Our private key (which is known only to us) is used to sign certain TLS/SSL handshake messages and to decrypt messages that are encrypted by remote peers by using our public key (which they find in the personal certificate that we send during the TLS/SSL handshake).
- Storage of certificates that we trust that are used to verify the authenticity of the remote peer's personal certificate during a TLS/SSL handshake (CA certificates and self-signed peer certificates).

Like RACF, IBM GSKit key fulfills both of these purposes by using a single repository that is called a *key database*. Configuration of the IBM GSKit key database is as critical to the Db2 TLS/SSL configuration on Linux, UNIX, and Windows as key ring configuration is to the z/OS TLS/SSL configuration.

Configuring TLS/SSL support in a Db2 Connect server for Linux, UNIX, and Windows

In this example, we illustrate how to configure the Db2 Connect server to support TLS/SSL to create a secure connection between the remote client and the server. In practice, the connection between the Db2 Connect server and the Db2 for z/OS server also should be protected by using AT-TLS.

In a Db2 Connect server for Linux, UNIX, and Windows environment, use the IBM GSKit to administer digital certificates and key databases. The IBM GSKit is installed automatically on the computer system during the installation of the Db2 Connect server. If you do not have the IBM GSKit installed, you can install the IBM GSKit libraries from an image that you downloaded from Passport Advantage.

Before we can proceed, ensure that the following environment settings are configured:

- On Windows (32-bit and 64-bit) platforms, verify that the path to the IBM GSKit libraries is included in the **PATH** environment variable. For example, on Windows, add the IBM GSKit `bin` and `lib` directories to the **PATH** environment variable:

  ```
  set PATH="C:\Program Files\IBM\gsk8\bin";"C:\Program Files\IBM\gsk8\lib";%PATH%
  ```

  On Linux and UNIX platforms, the IBM GSKit libraries are in `sqllib/lib` or `sqllib/lib64`. So, to set the **LIBPATH**, **SHLIB_PATH**, or **LD_LIBRARY_PATH** environment variables, you issue the following commands:

  ```
  export LIBPATH=$INSTHOME/sqllib/lib:$LIBPATH:.
  export SHLIB_PATH=$INSTHOME/sqllib/lib:$SHLIB_PATH:.
  export LD_LIBRARY_PATH=$INSTHOME/sqllib/lib:$LD_LIBRARY_PATH:.
  ```

- Ensure that the connection concentrator feature is disabled. TLS/SSL support is not enabled in the Db2 instance if the connection concentrator feature is enabled.

  To determine whether the connection concentrator feature is enabled, issue the **GET DATABASE CONFIGURATION** command. If the **max_connections** configuration parameter is set to a value greater than the value of **max_coordagents**, the connection concentrator feature is enabled.

To configure TLS/SSL support for a Db2 Connect server, you must create a server key database to manage the digital certificates. Next, the Db2 instance owner must configure the Db2 instance for TLS/SSL support. Complete the following steps:

1. Create a server key database and the associated digital certificates by using the **GSKCapiCmd** tool:

   a. Use the **GSKCapiCmd** tool to create a server key database of the type CMS. The **GSKCapiCmd** tool is a non- Java -based CLI tool. Java does not need to be installed on your system to use this tool.

   The path to the **GSKCapiCmd** tool is `C:\Program Files\IBM\gsk8\bin` on a 32-bit Windows platform. On 64-bit Windows platforms, the IBM GSKit executable files and libraries are also present, in which case the path for the command is `C:\Program Files (x86)\IBM\GSK8\bin`, and `sqllib/gskit/`bin on Linux and UNIX platforms.

   For example, to create a key database that is called `myserver.kdb` and a stash file that is called `myserver.sth`, issue the following command:

   ```
   gsk8capicmd -keydb -create -db "myserver.kdb" -pw "myServ3rPwd" -type cms
   -stash
   ```

   Specifying the **-stash** option creates a stash file for storing the password to the key database after creation. A stash file is an automatic way of providing the password to access the key database. The password is also encrypted when stored in the stash file so that when access to the key database is required, the system decrypts the password from the stash file and uses it as input to access the key database.

   b. Add or create a server certificate and private key in the server key database that we created in step a. This certificate is used by the remote clients to authenticate the server during the TLS/SSL handshake phase. You can obtain a certificate by using the **GSKCapiCmd** tool to create a certificate request and submit it to a CA to be signed, or you can create a self-signed certificate for testing purposes.

For example, to create a key pair and self-signed certificate with a certificate label of "myselfsigned", a key size of 2048 bits, and the signature algorithm SHA256withRSA, run the following command:

```
gsk8capicmd -cert -create -size 2048 -sigalg SHA256withRSA -db
"myserver.kdb" -pw "myServ3rPwd" -label "myselfsigned" -dn
"CN=myhost.mydomain.com,O=myOrganization,OU=myOrganizationUnit,L=myLocation,
ST=myState,C=myCountry"
```

c. Extract the server certificate from the server key database and distribute this certificate to all the remote clients that connect to this server by using TLS/SSL.

   For example, to extract the server self-signed certificate that you created from step b on page 60, run the following command:

```
gsk8capicmd -cert -extract -db "myserver.kdb" -pw "myServ3rPwd" -label
"myselfsigned" -target "myserver.arm" -format ascii -fips
```

   The extracted certificate is saved in Base64-encoded form in the file that is called `myserver.arm`. Because Base64-encoding is used, the file must be transferred to remote clients in text (ASCII) mode, not as binary data.

2. Configure the Db2 Connect server for TLS/SSL support. You might need to log in as the Db2 instance owner before you can set the SSL configuration parameters and DB2COMM registry variables.

   a. For example, specify the fully qualified path to the server key database in the **SSL_SVR_KEYDB** database configuration parameter:

```
db2 update dbm cfg using SSL_SVR_KEYDB C:\SSLServer\myserver.kdb
```

   b. For example, specify the fully qualified path to the server key database stash file in the **SSL_SVR_STASH** database configuration parameter:

```
db2 update dbm cfg using SSL_SVR_STASH C:\SSLServer\myserver.sth
```

SSL_SVR_STASH: If the **SSL_SVR_STASH** database configuration parameter is `NULL` (unspecified), SSL support is disabled.

   c. For example, specify the label of the server certificate in the **SSL_SVR_LABEL** database configuration parameter:

```
db2 update dbm cfg using SSL_SVR_LABEL myselfsigned
```

SSL_SVR_LABEL: If the **SSL_SVR_LABEL** database configuration parameter is `NULL` (unspecified), the default certificate label in the server key database is used. If there is no default certificate label in the server key database, SSL support is disabled.

   d. Specify, as an example, the secure port number for the Db2 Connect server to accept secure connections in the **SSL_SVCENAME** database configuration parameter:

```
db2 update dbm cfg using SSL_SVCENAME 50088
```

Important: If the **DB2COMM** registry variable specifies both TCP/IP and SSL (that is, `DB2COMM=TCPIP,SSL`), the port number that you specify for the **SSL_SVCENAME** must differ from the port number that is associated with **SVCENAME**. If the port number that you specify for the **SSL_SVCENAME** is the same as the port number for the **SVCENAME**, TCP/IP and SSL support are *not* enabled. Also, if the `SSL_SVCENAME` is set to `NULL` (unspecified), SSL support is disabled.

e. (Optional) You can specify the set of cipher suites for the server to use for TLS/SSL by setting the **SSL_CIPHERSPECS** database configuration parameter. The list of suites should conform to your local TLS standards. If you do not specify a list of suites, then IBM GSKit automatically picks the strongest cipher suite that is common to the server and the client.

f. Set the SSL communication protocols for the Db2 instance in the **DB2COMM** registry variable. For example, use the **db2set** command to set **DB2COMM** with the value of `TCPIP,SSL`:

```
db2set -i db2instdef DB2COMM=TCPIP,SSL
```

In this example, `db2instdef` is the Db2 instance name. If you want to start Db2 with only SSL support, you can set `DB2COMM=SSL`.

g. Restart the Db2 instance:

```
db2stop
db2start
```

Configuring Java or non- Java applications to use TLS/SSL to access the Db2 Connect server

To configure Java applications by using the IBM DS Driver for JDBC and SQLJ (Type 4 connections) to use TLS/SSL, see “Configuring connections under the IBM DS Driver for JDBC and SQLJ to use TLS/SSL” on page 49.

To configure non- Java Db2 clients for Linux, UNIX, and Windows to use TLS/SSL, see “Configuring the IBM DS Driver non-Java interfaces: Command-line interface, ODBC, and .NET” on page 54.

For either type of client, you must import the server’s certificate that you created, the `myserver.arm` file, to your client keystore or key database. Transfer the file from the server to the client in text mode. You also must specify the correct port number of the server that accepts secure connections.

Client access to Db2 by using TLS/SSL client authentication

With Db2 for z/OS, a remote client application can access a Db2 for z/OS server by using the TLS/SSL client authentication model, which is described in “Coding the AT-TLS policy rules for TLS/SSL client authentication” on page 22 and “Registering a client certificate with RACF (optional)” on page 40.

The Db2 for z/OS server can perform advanced client authentication checking, which builds on the basic TLS/SSL client authentication model. With this support in place, the Db2 for z/OS server uses the z/OS user ID that is associated with the client’s digital certificate, in addition to the user ID that is provided during the DRDA handshake, to decide whether to allow the DRDA connection.

Before we proceed with the steps to configure the client and server to use the client login by using digital certificates, we briefly explore the Db2 for z/OS processing to authenticate a remote client accessing Db2 by using digital certificates.

Processing client access by using digital certificates at a Db2 for z/OS server

The Db2 for z/OS server completes a sequence of authentication tasks when handling a remote client connection request, as shown in Figure 10.

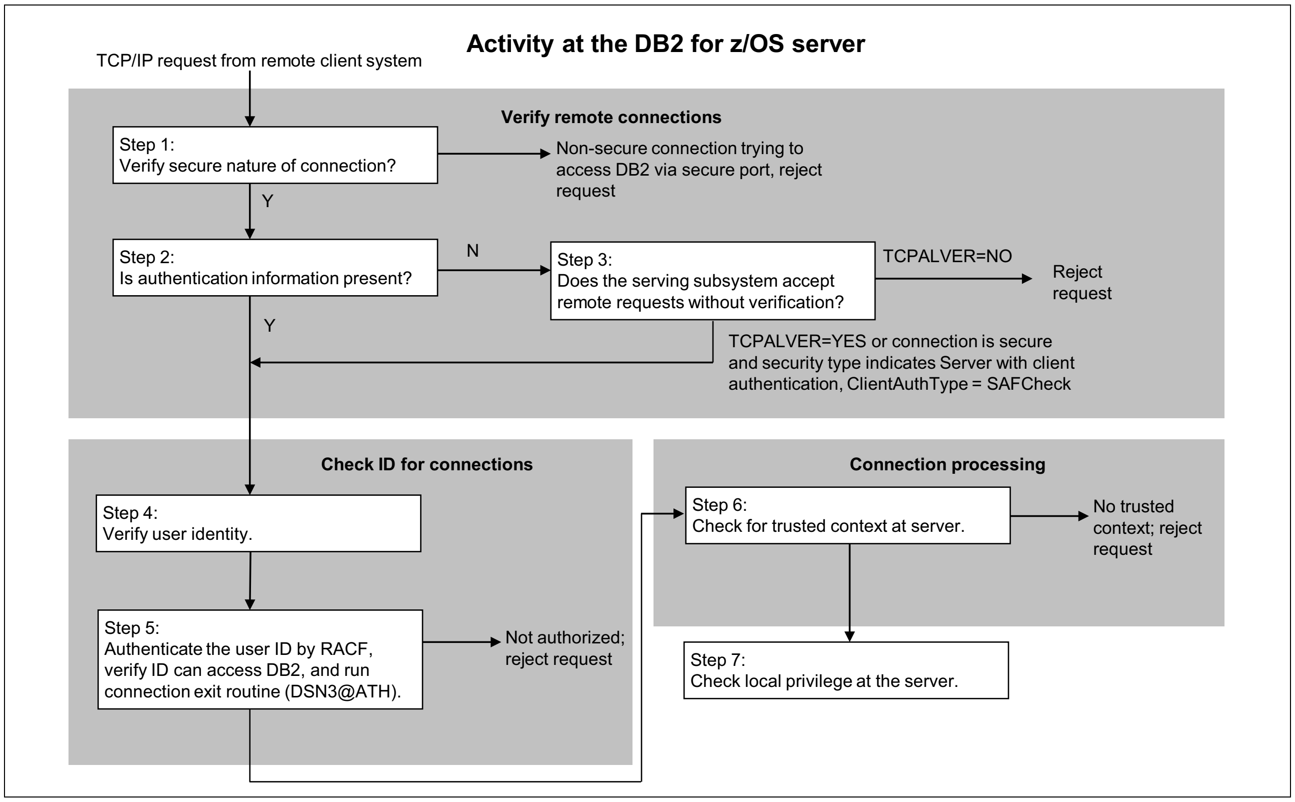

Activity at the DB2 for z/OS server
TCP/IP request from remote client system
Verify remote connections
Step 1: Verify secure nature of connection?
Non-secure connection trying to access DB2 via secure port, reject request
Y
Step 2: Is authentication information present?
N
Step 3: Does the serving subsystem accept remote requests without verification?
TCPALVER=NO
Reject request
Y
TCPALVER=YES or connection is secure and security type indicates Server with client authentication, ClientAuthType = SAFCheck
Check ID for connections
Step 4: Verify user identity.
Step 5: Authenticate the user ID by RACF, verify ID can access DB2, and run connection exit routine (DSN3@ATH).
Not authorized; reject request
Connection processing
Step 6: Check for trusted context at server.
No trusted context; reject request
Step 7: Check local privilege at the server.

Figure 10 Processing of an inbound TCP/IP connection request at a Db2 for z/OS server

The following numbers correspond to the steps in Figure 10:

1. If the connection is accepted to a secure port, TLS/SSL protection of the connection must be verified. For Db2 to determine whether the connection is secure, Db2 issues the **`AT-TLS SIOCTTLSCTL`** IOCTL command to query the AT-TLS policy information for the connection:

   – If the IOCTL returns a policy status of `TTLS_POL_ENABLED` and a connection status of `TTLS_CONN_SECURE`, a matching policy rule is found, and TLS/SSL is enabled for the connection. Furthermore, if the policy type is configured with the **`HandshakeRole`** parameter set to `ServerWithClientAuth` and the **`ClientAuthType`** parameter is set to `SAFCheck`, the IOCTL also returns the UID that is associated with the client certificate, which is registered with RACF.

   – If the IOCTL returns a policy status of `TTLS_POL_NO_POLICY` (no matching policy rule is found for the connection) or `TTLS_POL_NOT_ENABLED` (a matching policy rule policy is found, but it is not enabled for the connection), the connection is not secure.

   If the connection is not secure and the remote connection is trying to access Db2 by using the secure port, Db2 rejects the connection request. Otherwise, Db2 enables the connection long enough to perform the next steps.

2. Db2 checks to see whether the client sent an authentication token (RACF-encrypted password, RACF PassTicket, DRDA-clear text password, DRDA-encrypted password, or Kerberos ticket) during the DRDA handshake, which flows immediately after the TLS/SSL handshake completes.

3. If no authentication token is supplied, Db2 checks the **`TCPALVER`** subsystem parameter to see whether Db2 accepts IDs without authentication information:
   - If `TCPALVER=NO` or `TCPALVER=SERVER`, Db2 requires the minimum of a UID and a password.
   - If `TCPALVER=SERVER_ENCRYPT`, Db2 requires a UID and a password. In addition, Db2 requires that the security credentials are AES-encrypted or that the connection is accepted on a port that ensures AT-TLS policy protection, such as a Db2 Security Port (SECPORT). Kerberos tickets are accepted. RACF PassTickets or nonencrypted security credentials are accepted only when the connection is secured by the IP network.
   - If `TCPALVER=YES` or `TCPALVER=CLIENT`, Db2 accepts TCP/IP connection requests that contain only a UID.
   - If the connection is secure, as described in step 1 on page 63, and the UID that is associated with the client certificate is known, Db2 also accepts TCP/IP connection requests that contain only a UID.
4. Db2 calls RACF to verify the UID. The UID is one of the following values:
   - The ID is a Kerberos principal that is validated by the Kerberos Security Server if a Kerberos ticket is provided.
   - The ID is a RACF UID that is authenticated by RACF together with the password or PassTicket, if provided.
   - The ID is a distributed ID if the presence of a distributed registry name is provided by the connection. When a distributed registry name is provided, the RACF UID is derived from a distributed name filter that is defined in RACF.

     To use distributed name filters in this situation, Db2 starts the RACF distributed identity propagation function to retrieve the RACF UID that is associated with the distributed identity and registry name. To use distributed identity propagation, ensure that you define a distributed identity filter, which maps the distributed identity and the distributed registry name with a RACF UID.

     If RACF cannot find a distributed identity name filter for the distributed identity and registry name pair, Db2 treats the distributed identity as a RACF UID and starts RACF to verify it together with the password, if provided.
5. The UID is verified and authenticated in the following manner:
   - When Kerberos tickets are used, the RACF UID is derived from the Kerberos principal identity. To use Kerberos tickets, ensure that you map Kerberos principal names with RACF UIDs.
   - When the remote client accesses Db2 by using a secure connection and a RACF UID is associated with the client certificate, Db2 compares the remote client UID with the certificate UID.
   - Assuming that the remote client UID and the UID that is associated with the client certificate differ, extra authentication processing by Db2 is required to authenticate the remote client accessing Db2 by using a certificate.

     Trusted connection: The DRDA UID takes precedence over the certificate UID. Db2 uses the UID that is obtained from the certificate to search for a trusted context. If it exists, Db2 processes the DRDA UID as a trusted context switch-user request.

   - If the remote client UID and the UID that is associated with the client certificate are the same, Db2 authenticates the UID (and password, if provided) with RACF.

Extra authentication is required to authenticate the remote client connection when the remote connection request accesses Db2 by using a digital certificate and the remote client UID differs from the RACF UID that is associated with the client certificate. Db2 performs more steps:

- The UID that is associated with the client certificate is authenticated by RACF. This UID is also verified by RACF to ensure that it can access Db2 and that the connection exit routine started.
- Assuming that the authentication is successful, Db2 searches for a matching trusted context that is defined for the UID that is associated with the client certificate. If the trusted context exists, an implicit trusted connection is now established between the remote connection request and the Db2 server.

  When the trusted connection is established, Db2 authenticates the remote client UID that is associated with the remote connection request. This UID also is verified by RACF to ensure that it can access Db2 and that the connection exit routine started.

  Assuming that the authentication is successful, Db2 determines whether the primary UID is enabled to use the trusted connection. If the trusted connection can be used, the remote client is allowed access into Db2 by using a digital certificate.

  If authentication is unsuccessful for the remote client UID or the remote client UID is not allowed to use the trusted connection, the remote connection request is rejected.
- If authentication is unsuccessful for the client certificate UID or a trusted connection cannot be established for the client certificate UID, the remote connection is rejected.

If special consideration to authenticate the remote client connection request is not required, the UID together with the password or RACF PassTicket, if provided, is authenticated by RACF. Db2 calls RACF to check the UID's authorization against the `ssnm.DIST` resource (in the DSNR resource class) and starts the Db2 connection exit routine (`DSN3@ATH`). The parameter list that is passed to the routine describes where the remote request originated.

In addition, depending on your RACF environment, the following RACF checks might also be performed. If the RACF APPCPORT class is active, RACF verifies that the ID is authorized to access z/OS from the port of entry (POE).

The POE that RACF uses in the **`RACROUTE VERIFY`** call depends on whether all the following conditions are true:

- The IP network access control is configured.
- The RACF SERVAUTH class is active.

If all these conditions are true, RACF uses the remote client's POE security zone name that is defined in the IP network access control definitions in the TCP/IP profile. If one or more of these conditions are not true, RACF uses the literal string `TCPIP`.

If this request is to change a password, the password is changed.

6. The remote request has a primary authorization ID, possibly one or more secondary IDs, and an SQL ID. (The SQL ID cannot be converted.) If the remote connection request originated from a Db2 for z/OS client, the plan or package owner ID also accompanies the request. Privileges and authorities that are granted to those IDs at the Db2 server govern the actions that the request can take.

Db2 searches for a matching trusted context. If Db2 finds a matching trusted context, it validates the following attributes:

- If the **SERVAUTH** attribute is defined for the identified trusted context and TCP/IP provides a RACF SERVAUTH profile name to Db2 during the establishment of the connection, Db2 matches the SERVAUTH profile name with the **SERVAUTH** attribute value.
- If the **SERVAUTH** attribute is not defined or the SERVAUTH name does not match the **SERVAUTH** that is defined for the identified trusted context, Db2 matches the remote client's TCP/IP address with the **ADDRESS** attribute that is defined for the identified trusted context.
- If the **ENCRYPTION** attribute is defined, Db2 validates whether the connection uses the proper encryption, as specified in the value of the **ENCRYPTION** attribute.
- If the **DEFAULT SECURITY LABEL** attribute is defined for the system authorization ID, Db2 verifies the security label with RACF. This security label is used for verifying multilevel security for the system authorization ID. However, if the system authorization ID is also in the **ALLOW USER** clause with **SECURITY LABEL**, that ID is used.

If the validation is successful, Db2 establishes the connection as trusted. If the validation is not successful, the connection is established as a normal connection without any additional privileges, Db2 returns a warning, and SQLWARN8 is set.

When the trusted connection is established, Db2 enables the trusted connection to be reused by a separate UID on a transaction boundary. Then, a sequence of tasks is performed by Db2 when the remote connection requests to switch the UID on a trusted connection:

- Db2 determines whether the primary authorization ID is allowed to use the trusted connection. If the **WITH AUTHENTICATION** clause is specified for the user, Db2 requires an authentication token for the user. The authentication token can be a password, a RACF PassTicket, or a Kerberos ticket.
- Assuming that the primary authorization ID is allowed, Db2 determines the trusted context for any **SECURITY LABEL** definition. If a specific **SECURITY LABEL** is defined for this user, it becomes the **SECURITY LABEL** for this user. If no specific **SECURITY LABEL** is defined for this user but a **DEFAULT SECURITY LABEL** is defined for the trusted context, Db2 verifies the validity of this **SECURITY LABEL** for this user through RACF by issuing the **RACROUTE VERIFY** request.

  If the primary authorization ID is allowed, Db2 performs a connection initialization. This connection initialization results in an application environment that truly mimics the environment that is initialized if the new user establishes the connection in the normal Db2 manner. For example, any open cursor is closed, and temporary table information is dropped.
- If the primary authorization ID is not allowed to use the trusted connection or if the **SECURITY LABEL** verification fails, the connection is returned to an unconnected state.

  The only operation that is allowed in this context is to establish a valid authorization ID to be associated with the trusted connection. Until a valid authorization is established and if any SQL statement is issued, an error (`SQLCODE -900`) is returned.

Configuring client access to Db2 by using TLS/SSL client authentication

To implement client access to a Db2 for z/OS server by using digital certificates, complete the following steps:

1. Configure the AT-TLS policy rules for the Db2 server to use TLS/SSL client authentication, as described in “Coding the AT-TLS policy rules for TLS/SSL client authentication” on page 22 and “Registering a client certificate with RACF (optional)” on page 40.
2. Optionally, create a trusted context definition for the client certificate RACF UID as the system authorization ID and allow any DRDA UID to be used by the trusted context.

Example 27 shows how to create a trusted context for the UID, USRT001, which is associated with the registered client certificate, and allow UIDs USRT020, USRT021, and ADMF002 to use this trusted context.

Example 27 Sample trusted context definition for client access to Db2 by using certificates

```
CREATE TRUSTED CONTEXT CTX1
BASED UPON CONNECTION USING SYSTEM AUTHID USRT001
ATTRIBUTES (ADDRESS '10.30.223.20')
NO DEFAULT ROLE
ENABLE
WITH USE FOR USRT020,USRT021,ADMF002;
```

Db2 is now ready to accept remote client access by using a digital certificate that is registered to RACF, which has a UID, USRT001, that is associated with it.

In Example 27, where a connection is established from an application server originating from IP address 10.30.223.20 that is authenticated with the certificate that is associated with user USRT001, the connection can switch the Db2 primary authorization ID to either UIDs USRT020, USRT021, or ADMF002 without requiring any credentials.

UID: With client access to Db2 that uses a digital certificate, the UID that the client application specifies for the connection must also be a RACF-defined UID. It does not have to be the same UID as the certificate UID, but the UID must be known to RACF and have access to Db2 resources (DSNR class).

In the case where the UID that is specified by the remote client application is a distributed UID, which is registered to a distributed registry name (the UID is not defined in RACF), you must configure Db2 support for z/OS identity name filters.

Configuring z/OS identity name filters for the Db2 server

A *distributed identity filter* is a RACF mapping association between a RACF UID and one or more distributed UIDs. You can use the RACF **RACMAP** command to associate a distributed UID with a RACF UID. RACF distributed identity filters are implemented through z/OS identify propagation. Distributed identity filters are supported on z/OS V1R11 and later releases.

Db2 supports z/OS identify propagation and distributed identity filters. You must create distributed identity filters to take advantage of this support.

To create a distributed identity filter, complete the following steps:

1. Activate the RACF general resource IDIDMAP class and enable it for RACLIST processing by running the following command:

   ```
   SETROPTS CLASSACT(IDIDMAP) RACLIST(IDIDMAP)
   ```

2. Define a distributed identity filter and associate the distributed username with a RACF UID by issuing the RACF **`RACMAP`** command. To define a filter for a non-LDAP username, specify the username as a simple character string to be defined in a non-LDAP registry. Suppose that the distributed username is ‘`MARY`’, which is defined in user registry ‘`Registry01`’. If you want to map this username to RACF UID “USRT021”, you can run the following command:

   ```
   RACMAP ID(USRT021) MAP -
   USERIDFILTER(NAME('MARY')) -
   REGISTRY(NAME('Registry01')) -
   WITHLABEL('Filter for MARY from Registry01')
   ```

3. Refresh the IDIDMAP class profile by running the following command:

   ```
   SETROPTS RACLIST(IDIDMAP) REFRESH
   ```

4. If necessary, review the distributed identity filter by running the following command:

   ```
   RACMAP ID(USRT021) LISTMAP
   ```

   If the new filter is successfully created, the following output is returned:

   ```
   Mapping information for user USRT021:
      Label: Filter for MARY from Registry01
      Distributed Identity User Name Filter:
         >MARY<
      Registry name:
         >Registry01<
   ```

The new filter assigns the RACF UID, USRT021, when the distributed identity is user MARY from Registry01. When user MARY authenticates her ID at her distributed application server and performs tasks that access a remote Db2 server system, Db2 passes distributed username MARY and registry name Registry01 as character strings to RACF.

During Db2 remote connection processing, Db2 calls the RACF **`RACROUTE REQUEST=VERIFY ENVIR=CREATE`** macro service. RACF uses these data values to search the IDIDMAP profiles for a matching filter. RACF finds the matching filter labeled “`Filter for MARY from Registry01`”and assigns it the USRT021 UID. Because the remote client accesses Db2 by using a digital certificate, Db2 performs a switch user with the trusted context, CTX1, to allow USRT021 to become the primary authorization ID for the connection.

The remote connection then runs its transactions with the authority of the USRT021 UID. Audit records for this transaction contain RACF UID USRT021, distributed user MARY, and registry name Registry01, which Db2 pass to RACF.

Using the Microsoft truststore

If the application is in a Windows environment and wants to use Microsoft Certificate Store (MSCS), also known as the Windows truststore) for authentication, you can use MSCS by setting up one of the following configurations:

- Using `javax.net.ssl` settings in your Java application.

  If you have access to the source code, the following system properties can be set in the application:

  ```
  System.setProperty("javax.net.ssl.trustStoreType","WINDOWS-ROOT");
  System.setProperty("javax.net.ssl.trustStore","NONE");
  ```

- Using `javax.net.ssl` settings as Java runtime options.

  If you have access to the script that starts your application, you can add the properties as a Java runtime options:

  ```
  java -Djavax.net.ssl.trustStoreType=WINDOWS-ROOT
  -Djavax.net.ssl.trustStore=NONE <java application to run>
  ```

- Using JCC driver settings.

  JCC properties for SSL can be configured to use the MSCS. Parameters can be set by using the DataSource or global properties file or though a URL:

  - Setting the property by using the DataSource method:

    ```
    ds.setSslTrustStoreType("WINDOWS-ROOT");
    ds.setSslTrustStoreLocation("NUL");
    ```

  - Setting the property by using the global properties file method:

    ```
    db2.jcc.sslTrustStoreType=WINDOWS-ROOT
    db2.jcc.sslTrustStoreLocation=NUL;
    ```

  - Setting the property by using the URL method:

    ```
    sslTrustStoreType=WINDOWS-ROOT;sslTrustStoreLocation=NUL;
    ```

Using the Windows keystore

If the application is running in a Windows environment and it wants to use the Windows keystore for SSL authentication, you can use the keystore by setting up one of the following configurations.

- Using `javax.net.ssl` settings in your Java application.

  If you have access to the source code of your application, you can set the following Java properties within your application:

  ```
  System.setProperty("javax.net.ssl.keyStoreType","Windows-MY");
  System.setProperty("javax.net.ssl.keyStore","NONE");
  ```

- Using `javax.net.ssl` settings as the Java runtime options by using the CLI.

  If you have access to the script that starts your Java application or you want to run your Java application by using the CLI, you can use the **`-D`** option to set the Java properties:

  ```
  -Djavax.net.ssl.keyStoreType=Windows-MY -Djavax.net.ssl.keyStore=NONE <java
  application to run>
  ```

- Using JCC driver settings.

  JCC properties for SSL can be configured to use the MSCS. Parameters can be set either through the DataSource, global properties, or the connection URL:

  – Setting the JCC property by using the DatasSource:

    ```
    ds.setSslKeyStoreType("Windows-MY");
    ds.setSslKeyStoreLocation("NUL");
    ```

  – Setting the JCC property by using the global properties file:

    ```
    db2.jcc.sslKeyStoreType=Windows-MY;
    db2.jcc.sslKeyStoreLocation=NUL;
    ```

  – Setting the JCC property by using the URL:

    ```
    sslKeyStoreType=Windows-MY;sslKeyStoreLocation=NUL;
    ```

References

For more information, see the following resources:

- *DB2 9 for z/OS: Distributed Functions*, SG24-6952
- *IBM z/OS V2R1 Communications Server TCP/IP Implementation Volume 1: Base Functions, Connectivity, and Routing*, SG24-8096-00
- *IBM z/OS V2R2 Communications Server TCP/IP Implementation: Volume 2 Standard Applications*, SG24-8361
- *IBM z/OS V2R2 Communications Server TCP/IP Implementation: Volume 4 Security and Policy-Based Networking*, SG24-8363
- *Security Functions of IBM DB2 10 for z/OS*, SG24-7959
- *Db2 12 for z/OS Administration Guide,* SC27-8844-02
- *z/OS V2R4 Communications Server: IP Configuration Guide*, SC27-3650
- *z/OS V2R4 Communications Server: IP Configuration Reference*, SC27-3651
- *z/OS V2R4 Communications Server IP Diagnosis Guide*, GC27-3652
- *z/OS V2R4 Communications Server: IP Messages Volume 2 (EZB, EZD)*, SC27-3655
- *z/OS V2R4 Security Server RACF Command Language Reference*, SA23-2292
- *z/OS V2R4 Security Server RACF Security Administrator's Guide*, SA23-2289
- *z/OS V2R4 Cryptographic Services System Secure Sockets Layer Programming*, SC14-7495
- Db2 11.5 Java Applications development for IBM data servers:

  https://www.ibm.com/docs/en/db2/11.5?topic=applications-java
- Db2 11.5 Introduction to Db2 Call Level Interface and ODBC:

  https://www.ibm.com/docs/en/db2/11.5?topic=applications-cli-odbc
- IBM GSKit return codes:

  https://www.ibm.com/docs/en/txseries/8.1.0?topic=communications-gskit-return-codes

Authors

This paper was produced by a team of specialists working at the IBM Research® Triangle Park Laboratory in Research Triangle Park, NC and the IBM Silicon Valley Laboratory in San Jose, CA.

Chris Meyer is an IBM Senior Technical Staff Member and the network security architect for the z/OS operating system. He has over 35 years of experience developing IBM operating systems and security-related software products.

Derek Tempongko is an IBM Advisory Software Engineer on the Db2 for z/OS development team. He specializes in the area of the DDF component. He has over 18 years of experience with IBM and the Db2 database product.

Thanks to the following people for their contributions to this project:

Martin Keen
IBM Research Triangle Park, NC

Jim Pickel and Hugh Smith
IBM Silicon Valley Laboratory, CA

Tejaswini Papanna and Shilu Mathai
Rocket Software

Now you can become a published author, too!

Stay connected to IBM Redbooks

Notices

Trademarks

REDP-4799-02

ISBN 0738460281

Printed in U.S.A.