Redpaper

IBM Information Management Software

Chris Meyer
Paolo Bruni

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

Introduction

This IBM® Redpaper™ publication provides information about how to set up and configure IBM DB2® for z/OS® with Transport Layer Security (TLS), the modern version of Secure Sockets Layer (SSL). This configuration is accomplished 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 (CLI) 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 provided is applicable to DB2 11 for z/OS and DB2 10 for z/OS.

Although we use z/OS V2R1 as the reference release in this paper, the instructions are valid on releases as early as z/OS V1R11.

Throughout the paper, we reference z/OS Security Server, or IBM Resource Access Control Facility (IBM RACF®) in a variety of contexts. It should be understood that anywhere we mention RACF, it implies any System Authorization Facility (SAF) 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 using IBM DS Driver for JDBC and SQLJ to use TLS/SSL

Configuring the IBM DS Driver non-Java interfaces, such as CLI/ODBC and Microsoft .NET

Configuring remote client applications to use TLS/SSL through a DB2 Connect server for Linux, UNIX, and Windows

Client access to DB2 using TLS/SSL client authentication

This paper presents additional information to 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 earlier SSL specifications developed by Netscape Corporation for securing communications that use Transmission Control Protocol/Internet Protocol (TCP/IP) sockets. Unless specifically 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, please refer to one of the many TLS/SSL primers available on the Internet. Here is one that we have found to be very useful:


The TLS and SSL protocols are designed to run at the application level. 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 SSL and TLS protocols (SSL V2, SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 as of this writing) including a robust set of application programming interfaces (APIs) for z/OS C and C++ applications to use.

In an effort to make TLS/SSL more accessible to z/OS applications, z/OS Communications Server V1R7 introduced AT-TLS. AT-TLS starts TLS/SSL primitives in the TCP layer of the TCP/IP stack on behalf of application programs. based on policy files that describe the application traffic and how to protect it. With AT-TLS, z/OS applications written in almost any language 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 V2R1 Communications Server: IP Configuration Guide, SC27-3650.

Because AT-TLS is 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 start the required System SSL services to protect the data being transmitted over the network.
Figure 1 illustrates the flow of DB2 for z/OS acting as a server and AT-TLS, using TLS to protect the data exchanges over the network with a remote client system that also supports TLS.

The diagram in Figure 1 illustrates the following flow:

1. After the policy agent has successfully read the AT-TLS policy and installed it into the TCP/IP stack, the client's TCP connection to the DB2 server is established 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 AT-TLS policy and sees that AT-TLS protection is configured for this connection. As such, it 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 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, invokes System SSL to decrypt the data, and then delivers the cleartext inbound data to the DB2 server.

6. The DB2 server receives data in the clear.

Likewise, when the DB2 server sends data to the DB2 client, AT-TLS takes the cleartext outbound data, calls System SSL to encrypt it, and then sends the encrypted data to the client.
Configuring DB2 for z/OS as a server with TLS/SSL support

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

- The preferred approach is to use the z/OS Management Facility (z/OSMF) Configuration Assistant for z/OS Communications Server. The Configuration Assistant is a graphical user interface (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 panels in the Configuration Assistant tool. For a complete example of configuring AT-TLS using the Configuration Assistant, see “Chapter 12. Application Transparent Transport Layer Security” in IBM z/OS V2R1 Communications Server TCP/IP Implementation Volume 4: Security and Policy-Based Networking, SG24-8099.

- The alternative approach is to hand-code all of the necessary job control language (JCL), RACF directives, configuration files, and policy files. To provide the reader greater insight into the policy agent and AT-TLS configuration, this is the approach we describe in detail for the remainder of this document.

To configure DB2 for z/OS as a server with TLS/SSL support, perform the following steps:

1. Configure the policy agent as a started task in z/OS.
2. Define the security authorization for the policy agent.
3. Define the policy agent configuration files.
4. Configure AT-TLS.
5. Define the AT-TLS policy rules.
6. Create and configure digital certificates in RACF.
7. Configure a secure port for the DB2 for z/OS server.

We describe these steps in the following sections.

Configure the policy agent as a started task in z/OS

The policy agent 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 policy agent.

To start the policy agent as a z/OS started task, you can use the started procedure shown in Example 1.

Example 1   PAGENT JCL started procedure

```
//PAGENT   PROC
//PAGENT   EXEC PGM=PAGENT,REGION=0K,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 via STDENV.
//*
//STDENV   DD DSN=SYS1.TCPPARMS(PAENV),DISP=SHR
//*
//* Output 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. When the -d parameter is specified, however,
// 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™) 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 have configured the
environment variables for PAGENT in an MVS data set, SYS1.TCPPARMS, and 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 policy agent 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 PAGENT
     configuration file is a member of the SYS1.TCPPARMS data set in this
     example.

PAGENT_LOG_FILE  Specifies the log file name that is used by PAGENT. It is set to log
     policy agent messages to SYSLOGD.

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

Define the security authorization for the policy agent

The policies managed by the policy agent can affect system operation significantly. Therefore,
you need to restrict the list of z/OS user IDs under which policy agent is allowed to run. To do
this, you need to define certain resources and controls in the system’s security manager
product, such as RACF.

To set up the policy agent’s security definitions to RACF, perform the following steps:

1. Define the PAGENT started task to RACF.
2. Define the PAGENT user ID.
3. Associate the PAGENT user ID with the PAGENT started task.
4. Give authorized users access to manage the PAGENT started task.
5. Restrict access to the pasearch UNIX command.
Define the PAGENT started task to RACF

In this example, the policy agent runs under z/OS as a started task named PAGENT. To set up the policy agent started task to RACF, you need to define a profile for it to the RACF generic resource class called STARTED using the RDEFINE command.

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

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

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

```plaintext
//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, such as that shown in the following example:

```
RDEFINE  STARTED  SYSLOGD.*
```

Define the PAGENT user ID

In this example, the policy agent runs under the z/OS user ID PAGENT. Before z/OS V1R13, the user ID for the PAGENT started task must have z/OS UNIX superuser authority (the z/OS UNIX user ID (UID) for this user must be set to 0). Also, you need to assign a default group (DFLTGRP) for the user ID.

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

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

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

If you are using z/OS V1R13 and later, you can choose to use a non-superuser UID. See z/OS V2R1 Communications Server: IP Configuration Guide, SC27-3650 for more information on using such a UID.

If your security administrator has 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 user IDs.
If you are also going to log messages to SYSLOGD, define a user ID with superuser authority for the SYSLOGD started task, as shown in the following example:

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

**Associate the PAGENT user ID with the PAGENT started task**

Use the RACF `RALTER` command to associate the PAGENT user ID that was created from the prior step to the PAGENT started task, as illustrated in Example 5.

```
//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 need to associate the SYSLOGD user ID with the SYSLOGD started task, as shown in the following example:

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

**Give authorized users access to manage the PAGENT started task**

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

```
//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 
/*
```

**Restrict 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 shows the parsed results of the policy definition attributes. Note that the policy agent is designed to ignore unknown attributes, so misspelled attributes result in the default values being used.

The `pasearch` output can be used to verify that policies are defined correctly. However, you might not want every user to be able to see the policy definitions. To restrict unauthorized access to the `pasearch` command, a resource is defined to the SERVAUTH RACF resource class. This type of resource 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**: Policy type that is being requested. The following types are possible:
  - **QoS**: Policy Quality of Service
  - **IDS**: Policy intrusion detection system
  - **IPSec**: Policy Internet Protocol Security
  - **TLS**: Policy AT-TLS

**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 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 the USRT001 and USRT002 user IDs are permitted to use pasearch for the TCP/IP stack named TCPIP on the z/OS system named UTEC224.

**Example 7  RACF commands to restrict unauthorized users access to the pasearch UNIX command**

```plaintext
//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 have set up all of the security authorizations for the policy agent, you need to configure the policy for AT-TLS.

**Define the policy agent configuration files**

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

- Define the TcpImage statements
- Define the appropriate logging level
Define the TcpImage statements
The TcpImage statement specifies a TCP/IP image (a TCP/IP stack) and its associated image-specific configuration file. Each TcpImage statement can specify the file name of an image-specific configuration file. If this 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.

CommonTTLSConfig statement: The main configuration file optionally can contain a CommonTTLSConfig statement that identifies a shared AT-TLS policy file for all of the TCP/IP stacks in your environment.

Depending on your environment, you can define the TcpImage statement in the following way:

- If your environment is configured with multiple TCP/IP stacks, and each TCP/IP stack has its own policy definitions, specify TcpImage statements identifying an image-specific configuration file for each TCP/IP stack. See Figure 2 for an illustration of this type of configuration.

- If your environment is configured with multiple TCP/IP stacks, and all of the TCP/IP stacks use the same policy definitions, specify TcpImage statements for each 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 all the TCP/IP stacks.

Figure 2 Multiple TCP/IP stacks and multiple AT-TLS policies
Figure 3 shows an illustration of this type of configuration.

If your environment is configured with a single TCP/IP stack, only one AT-TLS policy definition exists. 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 4 shows an illustration of this type of configuration.

It is possible that TCP/IP stacks that are configured to the policy agent are not started or even defined. When this situation happens, the policy agent 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>
<thead>
<tr>
<th>Parameter</th>
<th>When used</th>
<th>Results</th>
</tr>
</thead>
</table>
| FLUSH and NOFLUSH | Used after policies are read without errors, when triggered by the following events:  
|                  | - Policy agent start-up  
|                  | - TcpImage statement added  
|                  | - MODIFY REFRESH command issued | For FLUSH:  
|                  | For NOFLUSH:  
|                  | - No policies are deleted from policy agent or the TCP/IP stack  
|                  | - Policies removed from the configuration file are not deleted from the policy agent or the TCP/IP stack | For PURGE:  
|                  | For NOPURGE:  
|                  | - No policies are deleted from the policy agent  
|                  | - No policies are deleted from the TCP/IP stack |
| PURGE and NOPURGE | - Policy agent shutdown  
|                  | - TcpImage statement deleted | |

When any (or all) TCP/IP stacks are shut down, the policy agent 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 a common AT-TLS policy file **SYS1.TCPPARMS(TTLSPOL)** to the TCP/IP image named TCPIP after flushing the existing policy control data when the TCPIP image is restarted:

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

**Define the appropriate logging level**

Use the LogLevel statement to control the amount of information logged by the policy agent. 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 information 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 policy agent. The following levels are supported:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SYSERR: System error messages</td>
</tr>
<tr>
<td>2</td>
<td>OBJERR: Object error messages</td>
</tr>
<tr>
<td>4</td>
<td>PROTERR: Protocol error messages</td>
</tr>
<tr>
<td>8</td>
<td>WARNING: Warning messages</td>
</tr>
<tr>
<td>16</td>
<td>EVENT: Event messages</td>
</tr>
<tr>
<td>32</td>
<td>ACTION: Action messages</td>
</tr>
<tr>
<td>64</td>
<td>INFO: Informational messages</td>
</tr>
<tr>
<td>128</td>
<td>ACNTING: Accounting messages</td>
</tr>
<tr>
<td>256</td>
<td>TRACE: Trace messages</td>
</tr>
</tbody>
</table>
To include more than one level of logging, specify the sum of all of 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 you might encounter during your initial AT-TLS setup. If this isn't sufficient, we advise you to set the LogLevel to 511. Be aware, however, that 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 file.

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 TCPIP TCP/IP image.

```
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 and Policy Agent when PAGENT is shut down normally.
  TcpImage TCPIP '/SYS1.TCPPARMS(TTLSPOL)' FLUSH PURGE
  # Here is an alternate way to specify a common AT-TLS policy for the
  # TCP/IP image:
  #
  # TcpImage TCPIP FLUSH PURGE
  # TTLSConfig '/SYS1.TCPPARMS(TTLSPOL)' FLUSH PURGE
  #
```

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

### Configure 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 policy agent.

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 specified in the AT-TLS action associated with the rule. If no such rule is found, the connection is not protected with TLS.
Set up TTLS stack initialization access control for AT-TLS

This step is optional. When AT-TLS is started during TCP/IP stack initialization, there might be a delay between the time that the stack comes up and when the policy agent 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 TCP/IP stack (TcpImage):

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

The variables have these definitions:

- **sysname**: The system name assigned to the z/OS LPAR.
- **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 policy agent is started. To do so, provide READ access to the EZB.INITSTACK.sysname.TcpImage profile in the SERVAUTH class for the z/OS user IDs under which such applications will run. All other applications are forced to wait until policy agent has initialized and its policies have been installed into the stack before connections are enabled.

Example 9 on page 15 shows the RACF commands that are used to enable the z/OS user IDs OMVSKERN, PAGENT, and SYSLOGD to establish TCP/IP connections before policy agent has loaded all of its policies. Note that this step is optional for DB2, because DB2 does not process any TCP/IP connections until the TCP/IP stack is completely initialized. By then, all applicable AT-TLS policies have been loaded into the TCP/IP stack.

**Tip:** To enable TTLS without modifying the 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 called TTLSON in the SYS1.TCPPARMS data set:
   
   `TCPCONFIG TTLS`

2. On the MVS console, issue this command:
   
   `VARY TCPIP,,0,DSN=SYS1.TCPPARMS(TTLSON)`

To disable TTLS, perform the following actions:

1. OBEYFILE called TTLSOFF in the SYS1.TCPPARMS data set:

   `TCPCONFIG NOTTLS`

2. On the MVS console, issue this command:

   `VARY TCPIP,,0,DSN=SYS1.TCPPARMS(TTLSOFF)`
Example 9  RACF commands to set up TCP/IP stack initialization access control for AT-TLS

```bash
//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  
/*
```

See the AT-TLS chapter in the z/OS V2R1 Communications Server: IP Configuration Guide, SC27-3650 for more information on using EZB.INITSTACK profiles, including advice on the minimum set of applications that will typically be given access.

AT-TLS configuration is complete. Next, we describe the commands to start and stop the PAGENT started task, and provide the commands to refresh policies in the policy agent.

**Starting and stopping policy agent from the z/OS console**

Use the MVS START command to start the policy agent 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, policy agent deletes all policies from the TCP/IP stack when policy agent shuts down.

**Define the AT-TLS policy rules**

An AT-TLS policy configuration file contains AT-TLS rules that identify specific types of TCP traffic, along with the type of TLS/SSL to be applied to those connections. If a rule match is found, AT-TLS transparently provides TLS protocol control for the connection that is based on the security attributes that are specified in the actions that are associated with the rule.

AT-TLS policy is matched (referred to as “policy mapping”) to a TCP connection the first time that 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 if a socket is readable or writable.
- A poll() to see if a socket is readable or writable.
- Any call that sends or receives data over a socket.
- An SIOCTTLSCTL IOCTL invocation, in the case of an 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. A variety of 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**: User ID of the owning process or wildcard user ID.
- **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**

<table>
<thead>
<tr>
<th>TTLSRule A</th>
<th>TTLSRule B</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>LocalAddr 1.1.1.1</td>
<td>LocalAddr 1.1.1.1</td>
</tr>
<tr>
<td>RemoteAddr 2.2.2.2</td>
<td>RemoteAddr 2.2.2.2</td>
</tr>
<tr>
<td>LocalPortRange 11000</td>
<td>LocalPortRange 11000</td>
</tr>
<tr>
<td>RemotePortRange 15000 16000</td>
<td>RemotePortRange 15500 16500</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Although both rules are identical in terms of IP addresses and local ports, they define different remote port ranges that actually 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 in the range 1 to 2,000,000,000, with 2,000,000,000 being the highest priority. When assigning priorities, skip several values to provide for future rule insertion between existing rules.

**Tip**: If a connection can map to more than one rule, always use priority, and leave priority space between rules.
When a rule match is found, policy lookup stops, and the connection is assigned the actions 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>
<thead>
<tr>
<th>Action reference</th>
<th>Action statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSGroupActionRef</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>TTLEnvironmentActionRef</td>
<td>N/A</td>
<td>This statement is only required 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.</td>
</tr>
<tr>
<td>TTLSConnectionActionRef</td>
<td>TTLSConnectionAction</td>
<td>This statement is optional.</td>
</tr>
</tbody>
</table>

For more information regarding these AT-TLS actions, see z/OS V2R1 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 additional measures that build upon one of those models.

In the most basic model, TLS/SSL provides server authentication wherein the 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 connecting to DB2.

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

In cases where proof of the client's identity is also important, TLS/SSL supports client authentication (sometimes called mutual authentication), which builds upon server authentication. With client authentication, when 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 is appropriate in situations where the server must ensure the identity of the clients that it serves. Client authentication is configured in AT-TLS policy by coding HandshakeRole ServerWithClientAuth on the TTLEnvironmentAction statement. When client authentication is used, AT-TLS supports several variations regarding the actual level of client authentication through the ClientAuthType parameter of the TTLEnvironmentAdvancedParms policy statement.
The most commonly used ClientAuthType values are:

<table>
<thead>
<tr>
<th>ClientAuthType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>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, handshake succeeds if System SSL successfully authenticates the certificate. Note that, 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.</td>
</tr>
<tr>
<td>SAFCheck</td>
<td>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 has been registered with RACF and mapped to a valid z/OS user ID before enabling a secure connection to be established. If the certificate is mapped to a z/OS user ID, the connection is enabled. If not, the connection is rejected.</td>
</tr>
</tbody>
</table>

When ClientAuthType SAFCheck is specified for IBM DB2 Distributed Relational Database Architecture™ (IBM DRDA®) connections, DB2 can add additional 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 mapped z/OS user ID. This query verifies that the user ID has permission to access the relevant \texttt{d2sn.DIST} resource in the RACF DSRN resource class (where \texttt{d2sn} is the DB2 subsystem name to which the client is connecting). If the z/OS user ID does not have permission to access the resource, the DRDA connection is dropped. This DB2 feature, called \textit{client certificate authentication}, enables you to restrict remote client access to the DB2 subsystem based on the client certificate mapping.

For a more detailed description of DB2 client authentication logic, see “Processing client access using digital certificates at a DB2 for z/OS server” on page 57.
Table 3 summarizes the DB2 client authentication options available with AT-TLS when the policy covering the DRDA connection specifies HandshakeRole ServerWithClientAuth.

<table>
<thead>
<tr>
<th>AT-TLS ClientAuthType</th>
<th>Client certificate</th>
<th>DSRN class is active and DB2 subsystem’s profile defined</th>
<th>Certificate validation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passthru</td>
<td>Optional</td>
<td>N/A</td>
<td>None.</td>
<td>Not recommended.</td>
</tr>
<tr>
<td>Full</td>
<td>Optional</td>
<td>N/A</td>
<td>If client certificate is provided, it is validated against server's key ring. If not, the connection is enabled.</td>
<td>Not recommended for DB2 DRDA connections.</td>
</tr>
<tr>
<td>Required</td>
<td>Required</td>
<td>N/A</td>
<td>Certificate is validated against server's key ring.</td>
<td>This is the default. Advised when additional z/OS user ID-based access controls are not required.</td>
</tr>
<tr>
<td>SAFCheck</td>
<td>Required</td>
<td>Required</td>
<td>Certificate is validated against server's key ring and must be associated with a user ID in the security product. In addition, DB2 adds client certificate authentication in certain cases depending on the user credentials specified in the DRDA handshake.</td>
<td>This is the strongest level of authentication. See “Processing client access using digital certificates at a DB2 for z/OS server” on page 57 for much more detail.</td>
</tr>
</tbody>
</table>

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

Because it is common for a single LPAR to support multiple DB2 for z/OS subsystems, we will illustrate the AT-TLS configuration and associated digital certificate configuration 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 using a method that makes it easy to add new DB2 for z/OS subsystems.

Each DB2 for z/OS subsystem is assumed to run under its own dedicated z/OS user ID. Your environment may be simpler (a single DB2 subsystem) or more complex (multiple subsystems, or scenarios where a subsystem needs to 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 show a simple AT-TLS policy that protects connections to our DB2A and DB2B subsystems.

Example 11  Simple AT-TLS policy for two similar DB2 for z/OS servers to support TLS/SSL connections

```plaintext
TTLRule DB2ASecureServer
{
  LocalPortRange 4801
  JobName DB2ADIST
  Direction Inbound
  Priority 1
  TTLGroupActionRef DB2@SecureGrpAct
  TTLSEnvironmentActionRef DB2@SecureServerEnvAct
}
TTLRule DB2BSecureServer
```
{  
  LocalPortRange 4802  
  JobName DB2BDIST  
  Direction Inbound  
  Priority 1  
  TTLSGroupActionRef DB2@SecureGrpAct  
  TTLSEnvironmentActionRef DB2@SecureServerEnvAct  
}

TTLSGroupAction DB2@SecureGrpAct
{
  TTLSEnabled On  
  FIPS140 Off  
  Trace 15  
}

TTLSEnvironmentAction DB2@SecureServerEnvAct
{
  HandShakeRole Server  
  TTLSKeyRingParmsRef DB2@KeyRing  
  TTLSCipherParmsRef DB2@CipherParms  
}

TTLSKeyRingParms DB2@KeyRing
{
  Keyring DB2@KEYRING  
}

TTLSCipherParms DB2@CipherParms
{
  # RSA, DSS auth with AES and SHA-2
  V3CipherSuites TLS_DHE_RSA_WITH_AES_256_CBC_SHA256  
  V3CipherSuites TLS_DHE_DSS_WITH_AES_256_CBC_SHA256  
  V3CipherSuites TLS_DH_RSA_WITH_AES_256_CBC_SHA256  
  V3CipherSuites TLS_DH_DSS_WITH_AES_256_CBC_SHA256  
  V3CipherSuites TLS_DH_RSA_WITH_AES_128_CBC_SHA256  
  V3CipherSuites TLS_DH_DSS_WITH_AES_128_CBC_SHA256  
  V3CipherSuites TLS_RSA_WITH_AES_256_CBC_SHA256  
  V3CipherSuites TLS_RSA_WITH_AES_128_CBC_SHA256  
  # RSA, DSS auth with AES and SHA-1
  V3CipherSuites TLS_DHE_RSA_WITH_AES_256_CBC_SHA  
  V3CipherSuites TLS_DHE_DSS_WITH_AES_256_CBC_SHA  
  V3CipherSuites TLS_DH_RSA_WITH_AES_256_CBC_SHA  
  V3CipherSuites TLS_DH_DSS_WITH_AES_256_CBC_SHA  
  V3CipherSuites TLS_DH_RSA_WITH_AES_128_CBC_SHA  
  V3CipherSuites TLS_DH_DSS_WITH_AES_128_CBC_SHA  
  V3CipherSuites TLS_RSA_WITH_AES_256_CBC_SHA  
  V3CipherSuites TLS_DH_RSA_WITH_AES_128_CBC_SHA  
  V3CipherSuites TLS_DH_DSS_WITH_AES_128_CBC_SHA  
  V3CipherSuites TLS_RSA_WITH_3DES_EDE_CBC_SHA  
  V3CipherSuites TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA  
  V3CipherSuites TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA  
  V3CipherSuites TLS_RSA_WITH_3DES_EDE_CBC_SHA  
  # RSA, DSS auth with 3DES and SHA-1
  V3CipherSuites TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA  
  V3CipherSuites TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA  
  V3CipherSuites TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA  
  V3CipherSuites TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA  
  V3CipherSuites TLS_RSA_WITH_3DES_EDE_CBC_SHA  
  V3CipherSuites TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA  
  V3CipherSuites TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA  
  V3CipherSuites TLS_RSA_WITH_3DES_EDE_CBC_SHA
}
The two TTLSRule statements define the rules that will protect the 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. Connection arrives on the respective local port (4801 and 4802).
3. The ports are owned by the DB2 distributed data facility (DDF) address spaces. DB2ADIST owns 4801 and DB2BDIST owns 4802.

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

The TTLSGroupActionRef parameter refers to the TTLSGroupAction statement named DB2SecureGrpAct, which is shared by both TTLSRule statements. This statement 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 DB2SecureGrpAct statement, a TTLSGroupAction statement named DB2SecureServerEnvAct, 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 TTLSGroupAction statement also includes a reference to a TTLSKeyringParms statement that defines a RACF key ring name, DB2KEYRING (key ring definitions are discussed in “Create and configure digital certificates in RACF” on page 30). Each DB2 for z/OS DDF address space requires its own key ring, which is qualified by its user ID. Because the AT-TLS policy only lists the key ring name without a qualifying user ID, it can be reused for 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 user ID. As such, DB2A will use DB2AUSER/DB2KEYRING and DB2B will use DB2BUSER/DB2KEYRING at run time.

The TTLSGroupAction also refers to the TTLSGroupAction statement that lists the cipher suites to be allowed. The TTLSGroupAction 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 TTLSGroupAction 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 shown:

(05) TLS_RSA_WITH_RC4_128_SHA
(04) TLS_RSA_WITH_RC4_128_MD5
(35) TLS_RSA_WITH_AES_256_CBC_SHA
(36) TLS_DH_DSS_WITH_AES_256_CBC_SHA
(37) TLS_DH_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
(30) TLS_DH_DSS_WITH_AES_128_CBC_SHA
(31) TLS_DH_RSA_WITH_AES_128_CBC_SHA
(32) TLS_DHE_DSS_WITH_AES_128_CBC_SHA
(33) TLS_DHE_RSA_WITH_AES_128_CBC_SHA
(0A) TLS_RSA_WITH_3DES_EDE_CBC_SHA
(16) TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
(13) TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
(10) TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
(0D) TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
(09) TLS_RSA_WITH_DES_CBC_SHA
There are a few important points to note here:

- This default list contains some cipher suites that are considered to be rather weak by today’s cryptographic standards, such as the use of single Data Encryption Standard (DES), MD5, and RC2, so we advise specifying a TTLSCipherParms statement that enforces a minimum cryptographic strength that meets your company’s security policies.

- If you need to comply with the US National Institute of Standards and Technology (NIST) Special Publication SP800-131a, you need to code a TTLSCipherParms statement that only includes compliant cipher suites (the one shown in Example 11 on page 19 complies with NIST SP800-131a, with preference given to the strongest suites).

- If you need to comply with NIST Special Publication SP800-52, as part of compliance with Federal Information Processing Standards Security Requirements for Cryptographic Modules, referred to as FIPS 140-2, you need to change TTLSRoupAction from FIPS140 Off to FIPS140 On. This ensures that DB2 for z/OS, when acting as a client, will only allow ciphers compliant with FIPS140-2 requirements.

Another reason you might need to specify the list of cipher suites is to match the capabilities of your communications partners. In order 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.

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

**TTLSEnvironmentAdvancedParms**

Contains a variety of advanced options that control various aspects of the TLS/SSL handshake and authentication, and control which TLS/SSL versions you want to use for the protected connections. By default, AT-TLS enables TLS V1.2, TLS V1.1, TLS V1.0, and SSL V3. For the purposes of ensuring FIPS 104-2 compliance, only TLS V1.1 and later can be enabled.

The TTLSEnvironmentAdvancedParms statement can be contained in or referenced by TTLSEnvironmentAction statements.

**TTLSSignatureParms**

Specifies details about the digital signature algorithms the you want to use to authenticate TLS/SSL communication partners. The TTLSSignatureParms statement can be contained in or referenced by TTLSEnvironmentAction or TLSConnectionAction statements.
TTLSConnectionAction
This action is optional, and is needed only when a subset of connections in an AT-TLS environment must have different parameters. TTLSConnectionAction serves the same purpose as the TTLSEnvironmentAction, but for a specific connection. When a TTLSConnectionAction statement is associated with the TTLSRule, its settings override those 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 a TTLSConnectionAction, its settings override those 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.

TTLSGskLdapParms
Specifies Lightweight Directory Access Protocol (LDAP) server parameters for certificate revocation list (CRL) checking. The TTLSGskLdapParms statement can be contained in or referenced by TTLSEnvironmentAction statements.

For more information on AT-TLS policy configuration, see the chapter about AT-TLS policy configuration and data protection in z/OS V2R1 Communications Server: IP Configuration Guide, SC27-3650.

For detailed syntax of each AT-TLS policy statement, see the chapter about AT-TLS policy statements, Policy Agent, and policy applications in z/OS V2R1 Communications Server: IP Configuration Reference, SC27-3651.

Coding the AT-TLS policy rules for TLS/SSL client authentication
If you need to verify the identity of each client that connects to DB2, use TLS/SSL client authentication. To use client authentication, 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

```sql
TTLSEnvironmentAction DB2ServerSecureEnvAct
{
    TTLSKeyRingParms
    {
        Keyring DB2@KEYRING
    }
    HandshakeRole ServerWithClientAuth
    TTLSCipherParmsRef DB2@CipherParms
    TTLSEnvironmentAdvancedParms
    {
        ClientAuthType SAFCheck
    }
}
```
Example 12 on page 23 specifies the HandshakeRole of ServerWithClientAuth, and defines a TTLSEnvironmentAdvancedParms statement to specify a client authentication type (ClientAuthType) of SAFCheck. This level of client authentication requires the client certificate to be registered with RACF, and a z/OS user ID to be associated with it in RACF (described in more detail in “Register a client certificate with RACF (optional)” on page 35.

Additionally, depending on how your DB2 subsystem obtains the client user credentials, this user ID might also need to be given permission to access the relevant d2sn.DIST resource in RACF, as described under “Peer authentication models” on page 17.

**Refreshing policies**

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

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

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

- **F PAGENT,UPDATE**  
  The UPDATE command triggers the policy agent to reread and process the policy files. This command differs slightly from the REFRESH command, because PAGENT only installs or removes from the stack any new, changed, or deleted policies, as appropriate (any unchanged policies are unaffected). Therefore, we advise that you use this command in most cases.

**Verifying policies**

Use the **z/OS UNIX pasearch** command to query information from the policy agent. 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, as shown in the following example:

```
oshell pasearch -t
```

*Example 13  Display AT-TLS policies using pasearch -t command*

<table>
<thead>
<tr>
<th>TCP/IP pasearch CS V2R1</th>
<th>Image Name: TCPCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date: 10/01/2014</td>
<td>Time: 21:36:00</td>
</tr>
<tr>
<td>TLS Instance Id: 1412213721</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>policyRule:</th>
<th>DB2ASecureServer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule Type:</td>
<td>TLS</td>
</tr>
<tr>
<td>Version:</td>
<td>3</td>
</tr>
<tr>
<td>Status:</td>
<td>Active</td>
</tr>
<tr>
<td>Weight:</td>
<td>1</td>
</tr>
<tr>
<td>ForLoadDist:</td>
<td>False</td>
</tr>
<tr>
<td>Priority:</td>
<td>1</td>
</tr>
<tr>
<td>Sequence Actions: Don't Care</td>
<td></td>
</tr>
<tr>
<td>No. Policy Action:</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>policyAction:</th>
<th>DB2@SecureGrpAct</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionType:</td>
<td>TLS Group</td>
</tr>
<tr>
<td>Action Sequence:</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>policyAction:</th>
<th>DB2@SecureServerEnvAct</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionType:</td>
<td>TLS Environment</td>
</tr>
</tbody>
</table>
Action Sequence: 0

Time Periods:
Day of Month Mask:
First to Last: 11111111111111111111111111111111
Last to First: 11111111111111111111111111111111
Month of Yr Mask: 11111111111
Day of Week Mask: 111 (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

Local Port From: 4801
Local Port To: 4801
Remote Port From: 0
Remote Port To: 0
JobName: DB2ADIST
UserId:
Service Direction: Inbound


TTLS Action: DB2SecureGrpAct
Version: 3
Status: Active
Scope: Group
TTLSEnabled: On
CtraceClearText: Off
Trace: 15
FIPS140: Off
TTLSGroupAdvancedParms:
SecondaryMap: Off
SyslogFacility: Daemon

TTLS Action: DB2SecureServerEnvAct
Version: 3
Status: Active
Scope: Environment
HandshakeRole: Server
SuiteBProfile: Off
TTLSKeyringParms:
Keyring: DB2KEYRING
TTLSEnvironmentAdvancedParms:
SSLv2: Off
SSLv3: On
TLSv1: On
TLSv1.1: On
TLSv1.2: 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
TTLSCipherParms:
v3CipherSuites:
006B  TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
006A  TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
0069  TLS_DH_RSA_WITH_AES_256_CBC_SHA256
0068  TLS_DH_DSS_WITH_AES_256_CBC_SHA256
0067  TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
0040  TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
003F  TLS_DH_RSA_WITH_AES_128_CBC_SHA256
003E  TLS_DH_DSS_WITH_AES_128_CBC_SHA256
003D  TLS_RSA_WITH_AES_256_CBC_SHA256
003C  TLS_RSA_WITH_AES_128_CBC_SHA256
0039  TLS_DH_RSA_WITH_AES_256_CBC_SHA
0038  TLS_DH_DSS_WITH_AES_256_CBC_SHA
0037  TLS_DH_RSA_WITH_AES_256_CBC_SHA
0036  TLS_DH_DSS_WITH_AES_256_CBC_SHA
0035  TLS_RSA_WITH_AES_256_CBC_SHA
0034  TLS_DH_RSA_WITH_AES_128_CBC_SHA
0033  TLS_DH_DSS_WITH_AES_128_CBC_SHA
0032  TLS_RSA_WITH_AES_256_CBC_SHA
0030  TLS_DH_DSS_WITH_AES_128_CBC_SHA
002F  TLS_RSA_WITH_AES_128_CBC_SHA
001F  TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
001E  TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
001D  TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
001C  TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
001B  TLS_RSA_WITH_3DES_EDE_CBC_SHA
001A  TLS_DHE_RSA_WITH_AES_128_CBC_SHA
0019  TLS_DHE_DSS_WITH_AES_128_CBC_SHA
0018  TLS_DH_RSA_WITH_AES_128_CBC_SHA
0017  TLS_DH_DSS_WITH_AES_128_CBC_SHA
0016  TLS_RSA_WITH_AES_128_CBC_SHA
0015  TLS_DHE_RSA_WITH_AES_256_CBC_SHA
0014  TLS_DHE_DSS_WITH_AES_256_CBC_SHA
0013  TLS_DH_RSA_WITH_AES_256_CBC_SHA
0012  TLS_DH_DSS_WITH_AES_256_CBC_SHA
0011  TLS_RSA_WITH_AES_256_CBC_SHA

cPolicyRule: DB2BSecureServer
Rule Type: TLS
Version: 3  Status: Active
Weight: 1  ForLoadDist: False
Priority: 1  Sequence Actions: Don't Care
No. Policy Action: 2
policyAction: DB2@SecureGrpAct
ActionType: TLS Group
Action Sequence: 0
policyAction: DB2@SecureServerEnvAct
ActionType: TLS Environment
Action Sequence: 0
Time Periods:
Day of Month Mask: 11111111111111111111111111111111
First to Last: 11111111111111111111111111111111
Last to First: 11111111111111111111111111111111
Month of Yr Mask: 111111111111
Day of Week Mask: 111111 (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:

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

TTLS Action: DB2@SecureGrpAct
Version: 3
Status: Active
Scope: Group
TTLSEnabled: On
CtraceClearText: Off
Trace: 15
FIPS140: Off
TTLSGroupAdvancedParms:
SecondaryMap: Off
SyslogFacility: Daemon

TTLS Action: DB2@SecureServerEnvAct
Version: 3
Status: Active
Scope: Environment
HandshakeRole: Server
SuiteBProfile: Off
TTLSKeyringParms:
Keyring: DB2@KEYRING
TTLSEnvironmentAdvancedParms:
SSLv2: Off
SSLv3: On
TLSv1: On
TLSv1.1: On
TLSv1.2: 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
TTLSCipherParms:
v3CipherSuites:
006B  TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
006A  TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
0069  TLS_DH_RSA_WITH_AES_256_CBC_SHA256
Diagnosing AT-TLS problems

There are several useful tools for diagnosing AT-TLS problems. See the chapter about diagnosing AT-TLS in z/OS Communications Server IP Diagnosis Guide, GC27-3652, for a complete description of common AT-TLS start-up errors, debugging steps and tools, and descriptions of AT-TLS return codes. This document explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the TCP/IP product code. It 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 currently installed.
  - The TSO `Netstat` command and its z/OS UNIX counterpart, `netstat`, provides AT-TLS information as it relates to specific TCP connections. The `C0mn/-c` and `TTL5/-x` options are especially useful for AT-TLS diagnosis.

  See z/OS Communications Server V2R1 IP System Administrator’s Commands, SC27-3661, for a complete description of these commands and their output.

- **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) as specified in the `/etc/syslog.conf` file.

  AT-TLS messages begin with the prefix EZD. Most are in the range EZD1281I through EZD1290I. Descriptions of the messages, along with suggested actions, can be found in z/OS V2R1 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 1-4999 come directly from System SSL. These return codes, including detailed descriptions and suggested actions, are described in the chapter about messages and codes in the z/OS V2R1 Cryptographic Services System Secure Sockets Layer Programming, SC14-7495.

- Return code values 5000 and higher are generated by AT-TLS. These return codes, including detailed descriptions and suggested actions, are described in the chapter about diagnosing AT-TLS in the z/OS Communications Server IP Diagnosis Guide, GC27-3652.

For more information on configuring syslogd, see the section about configuring the syslog daemon in z/OS V2R1 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 statement. 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, you can enable System SSL tracing. For more information on setting up and using System SSL traces, see the chapter about obtaining diagnostic information in the z/OS V2R1 Cryptographic Services System Secure Sockets Layer Programming, SC14-7495. One important note, however: 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 shows a DB2 message that is often generated in the syslog when a Distributed Relational Database Architecture (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 generated in the db2diaglog on a non-Java client:

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

For a description of the possible IBM Global Security Kit for SSL (GSKit) return codes, see the following website:

Create and configure digital certificates in RACF

There are numerous ways to create and manage digital certificates and their related key pairs.

On z/OS, this includes 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.

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

There are also many facilities on other platforms, from command-line tools to complete digital certificate management systems, that can be used as well. When these tools are used, the certificates and keys they generate must be imported onto z/OS using either RACF or gskkyman.

If your company already 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 first experimenting with 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 document, 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 on gskkyman, see the chapter about certificate and key management in z/OS V2R1 Cryptographic Services System Secure Sockets Layer Programming, SC14-7495. For more information on z/OS PKI Services, see z/OS V2R1 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.
- Manage the 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 of the functions that it supports, see the chapter about RACF command syntax in z/OS V2R1 Security Server RACF Command Language Reference, SA23-2292.

The next sections describe the following topics:

- Using the RACDCERT command to administer digital certificates and key rings
- Creating the keys, digital certificates, and key ring using RACDCERT
- Register a client certificate with RACF (optional)
- Creating and activating client certificate name filters
Using the RACDCERT command to administer digital certificates and key rings

The RACDCERT command is your primary administrative tool for managing digital certificates and the key rings of stored digital certificates. 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 needs to be given to a user, unless the user has the SPECIAL attribute. One of the following authorities must be given 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 has other uses)

In “Coding the AT-TLS policy rules for TLS/SSL server authentication” on page 19, where we defined the AT-TLS rule DB2ASecureServer and DB2BSecureServer, one of the conditions specified was the JobName condition. The JobNames 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 only apply if a job with the specified name owns the TCP connection.

Because AT-TLS starts System SSL (which uses RACF digital certificate APIs) under the user ID of the application that owns the TCP connection, specifying the job name also means that AT-TLS and System SSL will be operating under the started task's user ID. As a result, we need to give sufficient authority to the user ID that is associated with the started task to access the IRR.DIGTCERT.function resource in the FACILITY class.

In our example, the user ID associated to the started task DB2ADIST is DB2AUSER, and the user ID associated with DB2BDIST is DB2BUSER.

We will also assume that a third separate user ID is used for the purpose of managing the certificates and keyrings in RACF. Our scenario enables multiple DB2 for z/OS subsystems for AT-TLS protection. Therefore, we will assume that all of the RACF commands used to create, export, and import certificates, and those to create and populate keyrings, are issued from a z/OS user ID that has authority to perform such operations on behalf of itself, other z/OS user IDs, and for certificate authority (CERTAUTH) certificates and keys.

As described above, this user ID must have CONTROL access to the IRR.DIGTCERT.LIST resource in the FACILITY class. For illustration purposes, we will call this user ID CERTMGR. In practice, this can be whatever userid you want to use. In a simple test environment, that only defines a single DB2 for z/OS subsystem, you may even choose to use the user ID associated with the DB2 started task.
Example 15 illustrates how to grant the appropriate authority to these user IDs.

**Example 15  Granting permission for RACF digital certificate resources**

```
//RACDCERT EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*  
//SYSUADS DD DSN=SYS1.UADS,DISP=SHR  
//SYSLBC  DD DSN=SYS1.BROADCAST,DISP=SHR  
//SYSTSIN DD *
* Activate required SAF classes and define required resources
SETROPTS CLASSACT(DIGTCERT DIGSTRING)
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 keyrings
* on behalf of DB2 user IDs and for certificate authorities
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(CERTMGR) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CERTMGR) ACCESS(READ)
* Activate all of the above changes
SETR RACLIST (DIGSTRING) REFRESH
SETR RACLIST (DIGTCERT) REFRESH
SETR RACLIST (FACILITY) REFRESH
/*
```

**Creating the keys, digital certificates, and key ring using RACDCERT**

We are now ready to generate the necessary keys and digital certificates. Note that all of the RACF commands are issued from the designated certificate management user ID (CERTMGR in our scenario), and that the commands explicitly specify the intended owner of those certificates and keyrings.

In this scenario, we use a local Certificate Authority (CA), which is the entity trusted by DB2 and the DB2 clients to assure 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/SSL handshake. This certificate is signed by using the CA’s private key, and includes the DB2 for z/OS subsystem’s public key.

   System SSL uses DB2’s private key to decrypt any encrypted TLS/SSL handshake messages sent by the client to DB2 for z/OS. The DB2 clients use the server’s public key (obtained from server certificate) to encrypt those messages.
Figure 5 illustrates the required private key and certificate configuration for this scenario, assuming 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 using DB2B’s own key ring, private key, and certificate, but with the same CA certificate (because we're using the same CA to sign all of the DB2 for z/OS personal certificates).

Figure 5   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 16. Note that CA certificates are not associated with any user ID, so the CERTAUTH keyword is specified instead of ID(userid).

Example 16   Generate a CA certificate

```c
RACDCERT CERTAUTH -
  GENCERT -
  SUBJECTSDN(OU('SVL224 CA') -
    O('IBM') -
    L('SVL') -
    SP('CA') -
    C('US')) -
  NOTAFTER(DATE(2030-12-31)) -
  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 previously, use the RACDCERT command that is shown in Example 17. Note that each key pair and certificate are owned by the user IDs 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, each owned by the user ID 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 will also set a reasonable certificate expiration date (this will need to be tracked carefully by the certificate administrator so that the certificate can be renewed well before the certificate expires).

Example 17  Generate key pairs and personal certificate for the DB2 server for z/OS subsystems

Example 18  Create the key rings and connect the digital certificates and private keys

Now that we have generated the CA and DB2 subsystem keys and certificates, we need to 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 user ID under which the respective DB2xDIST address space will run. Example 18 lists the RACDCERT commands to create the DB2@KEYRING key rings and hold the required digital certificates and private keys.
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 so that it can be used to verify the DB2 server certificate during the TLS/SSL handshake.

RACF supports a variety of export formats, including Distinguished Encoding Rules (DER), Public Key Cryptography Standards (PKCS) #12, and PKCS#7, either in DER-encoding or Base64 encoding. Example 19 shows the RACDCERT command to export the CA certificate 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, where those that are DER encoded should be transported as binary. Note that the default Base64-encoded DER takes the binary DER encoded data and then encodes it using Base64, so it should be transported as text.

Example 19  Exporting the CA certificate
RACDCERT CERTAUTH -
  EXPORT(LABEL('SVL224 CA Cert')) -
  DSN('USRT001.SVL224.CACERT')

At this point, all of the AT-TLS and certificate configuration 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 “Configure a secure port for the DB2 for z/OS server” on page 38.

Register a client certificate with RACF (optional)
If TLS/SSL client authentication is configured (HandshakeRole is ServerWithClientAuth), the server needs to 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 that which signed the server certificate, you have to 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 have already seen 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 will only illustrate this for the DB2A server, but the same approach can be used for DB2B or any other DB2 for z/OS subsystems that you may have defined in your environment.

In some cases, you might want to configure TLS/SSL sessions without involving a CA. The most common case is when you are first learning about certificates, or when you are first trying out certificate-based authentication in a test environment.
Another common case is when the two communicating parties are especially well known to each other and are trusted, eliminating the need for a trusted third party to assist in peer authentication (for example, two departments in the company that are in the same building or site). In situations like these, 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, this means that the communications partner is vouching for its own identity. Note that self-signed certificates are not generally recommended for production environments, or for any environment on which the two communicating partners do not belong to the same business entity. However, due to their popularity and simplicity of configuration, the illustrated configuration shows the DB2 for z/OS subsystem establishing secure DRDA sessions with a client that uses a self-signed certificate.

Figure 6 illustrates our previous certificate configuration with a self-signed client certificate added. During the TLS handshake, and after the client has 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, thereby assuring the client’s identity.

Because the client is using a self-signed certificate in this case, there is no CA involved, so the client’s personal certificate serves as the trusted authority. As such, a copy of the client certificate is added to the server’s key ring before any TLS handshakes are attempted.

![Figure 6 Adding a self-signed client certificate to our configuration](image-url)
Example 20 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 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 is also marked as trusted so that System SSL trusts it to verify the client certificate sent during the TLS/SSL handshake.

**Example 20  Add a copy of the self-signed client certificate to RACF as a trusted CA certificate**

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

Next, we need to add this self-signed client certificate to the appropriate server key rings. In this case, we will assume that the client will only connect to DB2A, so we need to connect the client certificate to DB2AUSER/DB2@KEYRING. Example 21 on page 37 shows you how to do this with 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 user ID DB2AUSER, under the label 'SVLC SSCert'.

Because the certificate is being connected as a CA certificate, we don’t specify an owning user ID for the certificate - instead, we simply specify CERTAUTH, which also implies the certificate’s usage (therefore, we do not have to specify USAGE(CERTAUTH)).

**Example 21  Connecting a self-signed client certificate to a key ring**

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

At this point, 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 many client certificates with one z/OS user ID 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 large number of client certificates to a limited number of user IDs, which helps you reduce administrative costs.

AT-TLS ClientAuthType SAFCheck and DB2 client certificate authentication processing (as described under “Peer authentication models” on page 17) use the z/OS user IDs mapped through certificate name filters.

Follow these steps to create and activate a certificate name filter:

1. Create a certificate name filter by issuing the RACDCERT MAP command, as shown in Example 22.

   **Example 22  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’s distinguished name (SDN) in the certificate. The filter associates the USRT001 user ID 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.

   Issue the following command to activate the SETROPTS RACLIST processing for the DIGTNMAP class:

   ```
   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. Issue the following command to refresh the DIGTNMAP class:

   ```
   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 user ID (in these examples, the user ID that AT-TLS and DB2 will find when ClientAuthType SAFCheck is specified).

**Configure a secure port for the DB2 for z/OS server**

With DB2 for z/OS, DDF is capable of (optionally) listening to a secondary secure port for inbound TLS/SSL connections. The DDF SQL TCP/IP Listener accepts unprotected (cleartext) 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 non-secure 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 assured 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 7.

<table>
<thead>
<tr>
<th>DSNTIP5</th>
<th>INSTALL DB2 - DISTRIBUTED DATA FACILITY PANEL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter data below:</td>
<td></td>
</tr>
<tr>
<td>1 DRDA PORT     ===&gt; 446 TCP/IP port number for DRDA clients. 1-65534 (446 is reserved for DRDA)</td>
<td></td>
</tr>
<tr>
<td>2 SECURE PORT ===&gt; 4801 TCP/IP port number for secure DRDA clients. 1-65534 (4801 is reserved for DRDA using SSL)</td>
<td></td>
</tr>
<tr>
<td>3 RESYNC PORT   ===&gt; 5001 TCP/IP port for 2-phase commit. 1-65534</td>
<td></td>
</tr>
<tr>
<td>4 TCP/IP ALREADY VERIFIED ===&gt; NO Accept requests containing only a User ID (no password)? YES or NO</td>
<td></td>
</tr>
<tr>
<td>5 EXTRA BLOCKS REQ ===&gt; 100 Maximum extra query blocks when DB2 acts as a requester. 0-100</td>
<td></td>
</tr>
<tr>
<td>6 EXTRA BLOCKS SRV ===&gt; 100 Maximum extra query blocks when DB2 acts as a server. 0-100</td>
<td></td>
</tr>
<tr>
<td>7 AUTH AT HOP SITE ===&gt; BOTH Authorization at hop site. BOTH or RUNNER.</td>
<td></td>
</tr>
<tr>
<td>8 TCP/IP KEEPALIVE ===&gt; 120 ENABLE, DISABLE, or 1-65534</td>
<td></td>
</tr>
<tr>
<td>9 POOL THREAD TIMEOUT ===&gt; 120 0-9999 seconds</td>
<td></td>
</tr>
</tbody>
</table>

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

Figure 7 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 could configure AT-TLS to protect the DRDA PORT and clients could connect to it using TLS/SSL. However, if you do this, DB2 is completely 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 value of the SECPORT secure port is the same as the value of PORT or RESPOR, 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 needs to be the same, just as the DRDA PORT for each member also needs to be the same. If each DB2 member specifies a unique secure port, the behavior is unpredictable. For instance, 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 needs to 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 only want to define a location alias name 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 need to 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 only listens for TLS/SSL inbound socket requests 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 need to configure specific IP addresses for each DB2 member (for example, 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). Next, you remove the specification of the IP addresses from the PORT statement in the 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 has been defined and AT-TLS has been enabled for the outbound connection.

Tip:
If you only want to define a location alias name for a DB2 member, you do not need to define a separate unique secure port for the location alias.
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 back 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 \texttt{Y} in the SECURE column. You also need to 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 \textit{and} the value of the SECURE column specifies \texttt{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 non-secure 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 non-secure 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 using TLS/SSL protection, as shown in Figure 8.

![Figure 8 LOCATION ALIAS consideration](image)

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 the use of the LOCATION ALIAS name feature is to use the DBALIAS name feature.
Therefore, using the same scenario, at the DB2 requester, you can define two rows in the SYSIBM.LOCATIONS table, as shown in Figure 9:

- A row that specifies the location name, and the non-secure DRDA port of the remote server
- Another row that specifies a separate location name, 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

![Figure 9 LOCATION DBALIAS name feature](image)

**Configure the policy agent and AT-TLS**

At the DB2 requester, the policy agent and AT-TLS need to be configured on the requester's z/OS system. Note that a single DB2 for z/OS subsystem can act as both a server and requester simultaneously, so we will build upon the sample policy from Example 11 on page 19 to illustrate how to add requester policy for DB2A.

**Define AT-TLS policy rules**

The steps to define the AT-TLS policy rules are the same as the steps that were described in the previous section for defining AT-TLS policy rules for the DB2 server role, with the exception of these TTLSRule conditions:

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

Example 23 provides a simple AT-TLS policy that can be added to the policy in Example 11 on page 19 to allow DB2A to initiate secure outbound connections to remote servers listening on port 4801.

*Example 23 AT-TLS policy statements to allow DB2 requester traffic from DB2A*

```plaintext
TTLSRule DB2ASecureRequester
{
  RemotePortRange 4801
  JobName DB2ADIST
  Direction Outbound
  TTLSGroupActionRef DB2@SecureGrpAct
  TTLSEnvironmentActionRef DB2@SecureClientEnvAct
}
TTLSEnvironmentAction DB2@SecureClientEnvAct
```
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. In this example, that means 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 Certificate Authority, and that that CA’s certificate has been exported into a data set named USRT050.RTP853.CACERT and transferred using FTP onto the local z/OS system 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 24 shows the RACDCERT commands to perform these tasks.

Example 24   Store and add 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 policy agent at the client system using the MVS MODIFY command, or use the MVS START command if the policy agent 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 will need a copy of DB2A’s CA certificate added to its key ring or trust store if it hasn’t already been added there. In this scenario, DB2A, as 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 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 for z/OS 10 (or later versions) server using TLS/SSL. Two configuration steps are necessary:

- Configure 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 need to briefly explain some terminology and concepts behind storing the relevant certificates and keys.
On z/OS, we have stored all of our digital certificates and private keys on a RACF key ring. There are actually two purposes being served by these key rings:

- Storage of one’s personal certificate and the private key 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 only known to us) is used to sign certain TLS/SSL handshake messages, and to decrypt messages that are encrypted by remote peers 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 the purpose of verifying the authenticity of the remote peer’s personal certificate during a TLS/SSL handshake (these may be certificate authority 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, configuration of the keystore and truststore are as critical to Java’s TLS/SSL configuration as key ring configuration is to z/OS TLS/SSL configuration.

### Configure 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.sslConnection property to true.

You can also set the DB2BaseDataSource.sslTrustStoreLocation 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.sslTrustStoreLocation, the javax.net.ssl.trustStore property is not used.

You can also set the DB2BaseDataSource.sslTrustStorePassword on a Connection or DataSource instance to identify the truststore password. Setting the sslTrustStorePassword property is an alternative to setting the Java javax.net.ssl.trustStorePassword property. If you set DB2BaseDataSource.sslTrustStorePassword, javax.net.ssl.trustStorePassword is not used.

The code sample in Example 25 shows how to set the sslConnection property on a Connection instance.

**Example 25  sslConnection property**

```java
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);
```
Configure the Java Runtime Environment to use TLS/SSL

Before you can use TLS/SSL with your JDBC and SQLJ applications, you need to 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 only works 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.

Configure 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 19 on page 35 into Base64-encoded DER format, we must perform the FTP GET command in ASCII mode.

   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`, type the following command:

   ```
   keytool -genkey -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`. For example, to import the server’s CA certificate from the file named `svl224.cacert` into the truststore that we created previously using the alias (analogous to the RACF certificate label) `svl224_ca`, use the following command:

   ```
   keytool -import -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.

   **Keytool interface:** In Java SE 6 (or later), the keytool interface has changed. The `-genkey` option has been renamed to `-genkeypair`.

   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`. For example, to import the server’s CA certificate from the file named `svl224.cacert` into the truststore that we created previously using the alias (analogous to the RACF certificate label) `svl224_ca`, use the following command:

   ```
   keytool -import -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.

   **Keytool interface:** In Java SE 6 (or later), the keytool interface has changed. The `-import` option has been renamed to `-importcert`.

   **Important:** Do not use BINARY mode transmission to get the certificate.
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`, type the following command:

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

3. If you have 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 `IDB2BaseDataSource.sslTrustStoreLocation` IBM DS Driver for JDBC and SQLJ 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 `DB2BaseDataSource.sslTrustStorePassword` IBM DS Driver for JDBC and SQLJ 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 named `testssl.java` and set the `javax.net.ssl.trustStore` system property to specify a truststore named `myTrustStore`, type 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");
```

**Configure the JRE to use TLS/SSL client authentication**

The following steps demonstrate how to configure the JRE to use TLS/SSL client authentication:

1. Perform all of the steps from “Configure the JRE to use TLS/SSL server authentication” on page 45.

2. Use the Java `keytool` command to generate a self-signed client certificate. For example, the following `keytool` command generates a new Rivest-Shamir-Adleman algorithm (RSA) key pair and self-signed certificate in the keystore named “myKeyStore” using an alias of “clnt_cert”:

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

Note that 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.
3. Export the client certificate using the Java `keytool` command. For example, use this command:

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

The output from this command produces a file called `client.ca`, which contains the client certificate in Base64-encoded format (the `-rfc` parameter causes this encoding format).

4. 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:

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

Figure 10 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
Certificate fingerprints:
  Signature algorithm name: SHA1withRSA
  Version: 3
```

Figure 10  Sample output from the `keytool -printcert` command

5. 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 “Register a client certificate with RACF (optional)” on page 35.

6. 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 DB2BaseDataSource.sslKeyStoreLocation IBM DS Driver for JDBC and SQLJ 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 DB2BaseDataSource.sslKeyStorePassword IBM DS Driver for JDBC and SQLJ property is set, its value overrides the `javax.net.ssl.keyStorePassword` property value.

---

**DB2 for z/OS: Configuring TLS/SSL for Secure Client/Server Communications** 47
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 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:

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

## Configuring the IBM DS Driver non-Java interfaces, such as CLI/ODBC and Microsoft .NET

Starting with DB2 version 9.5 fix pack 2, 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, to use TLS/SSL for communications with a DB2 for z/OS 10 (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

### 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 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 GSKit libraries from an image that you download from IBM Passport Advantage®.

For information about the IBM GSKit tool GSKCapiCmd, see the *GSKCapiCmd User's Guide*, which is available on the following web page:


You also need to 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 (V7 or V8) `bin` and `lib` directories to the `PATH` environment variable:

  ```
  set PATH=C:\Program Files\IBM\gsk7\bin;"C:\Program Files\IBM\gsk7\lib";%PATH%
  ```

- On Linux and UNIX platforms, the GSKit libraries are in `sql1ib/lib` or `sql1ib/lib64`. To set the `LIBPATH`, `SHLIB_PATH`, or `LD_LIBRARY_PATH` environment variable, you issue the following commands:

  ```
  export LIBPATH=$INSTHOME/sql1ib/lib:$LIBPATH:
  export SHLIB_PATH=$INSTHOME/sql1ib/lib:$SHLIB_PATH:
  export LD_LIBRARY_PATH=$INSTHOME/sql1ib/lib:$LD_LIBRARY_PATH:
  ```
Alternatively, if you want to use the IBM Key Management tool, gsk7ikm, you need to set the 
`JAVA_HOME` environment variable before you can start it. 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 DB2 for z/OS: Distributed Functions, 
SG24-6952.

The keys, certificates, and key database must always conform to FIPS 140 mode (specified 
by the `-fips` parameter on the `gsk7capicmd` command) when they are used for TLS/SSL 
communication.

**GSKit key database versus z/OS key rings**

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

- Storage of one's personal certificate and the private key 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 only known 
  to us) is used to sign certain TLS/SSL handshake messages and to decrypt messages 
  that are encrypted by remote peers 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 (certificate authority certificates 
  and self-signed peer certificates).

Like RACF, the GSKit key fulfills both of these purposes using a single repository called a key 
database. As we will see, configuration of the 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 using 
TLS/SSL, you need to perform the following tasks:

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 19 on page 35 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.
   ```

2. On the DB2 client computer, use the `gsk7capicmd` tool to create a client key database of 
   the certificate management system (CMS) type. The `gsk7capicmd` tool is a 
   non-Java-based command-line 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 
   “myClntPw0rd” of type CMS in the directory `$DB2PATH\security\keystore`, use the 
   following `gsk7capicmd` command:

   ```
   gsk7capicmd -keydb -create -db "mydbclnt.kdb" -pw "myClntPw0rd" -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. Use the `gsk7capicmd` command to add the server's CA certificate to the key database that was created in step 2.

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

```
gsk7capicmd -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, issue the following `gsk7capicmd` command:

```
gsk7capicmd -cert -details -db "mydbclnt.kdb" -pw "myClntPw0rd" -label "EC754 CA"
```

## Configuring the appropriate client configuration parameter to use TLS/SSL

If your DB2 client level is at version 9.7 or later, you do not need to create an SSL configuration file. DB2 version 9.7 introduced two new DB2 configuration parameters (`DB2_SSL_KEYSTORE_FILE` and `DB2_SSL_KEYRING_STASH_FILE`) to replace the SSL parameters (`DB2_SSL_KEYSTORE_FILE` and `DB2_SSL_KEYRING_STASH_FILE`) that were used for specifying the fully qualified path for the client key database, and for the stash file. In this case, you can skip to step 3 on page 51.

Otherwise, you need to create an SSL configuration file:

1. Create an SSL configuration file.

   Depending on the platform type, the SSL configuration file needs to be stored in a specific directory on the client system:
   - For Linux and UNIX: `$INSTHOME/cfg`
   - For Windows: `$INSTHOME/

   An example of `INSTHOME` on the Windows platform is a concatenation of the `$DB2INSTPROF/$DB2INSTDEF` environment variables:

   `C:\DOCUMENTS AND SETTINGS\ALL USERS\APPLICATION DATA\IBM\DB2\DB2COPY1\DB2`

   - Name the SSL configuration file, `SSLClientconfig.ini`

2. Define the following SSL parameters in the `SSLClientconfig.ini` file:

   - `DB2_SSL_KEYSTORE_FILE`: Specifies the fully qualified name of the key database that stores the server's CA certificate.
   - `DB2_SSL_KEYRING_STASH_FILE`: Specifies the fully qualified name of the stash file that contains the encrypted password to access the key database.

   The following example shows an `SSLClientconfig.ini` file:

   ```ini
   DB2_SSL_KEYSTORE_FILE=C:\Program Files\IBM\SQLLIB\security\keystore\mydbclnt.kdb
   DB2_SSL_KEYRING_STASH_FILE=C:\Program Files\IBM\SQLLIB\security\keystore\mydbclnt.sth
   ```
3. Set the appropriate connection strings or configuration parameters for your client application:
   a. CLP clients:
      i. For CLP clients, you need to issue 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 stlecn
         ```
      
      ii. If you use DB2 Version 9.7 or later, 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 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. Connection string parameters are only applicable to DB2 Version 9.7 or later clients.

      For example, if your CLI application uses the IBM Data Server Driver for ODBC and CLI, and the DB2 client level is Version 9.7 or later, use the connection string parameters to specify the client key database and stash file.

      ii. Call SQDriverConnect 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\sqlib\security\keystore\mydbclnt.kdb; Ssl_client_keystash=c:\program files\ibm\sqlib\security\keystore\mydbclnt.sth;"
         ```
For example, if your CLI application uses the IBM Data Server Client or IBM Data Server Runtime Client, and the DB2 client level is Version 9.7 or later, 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:

```
[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 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 Version 9.7 and later, you can also specify the fully qualified path for the client key database file (.kdb) and for the stash file (.sth) using the `SSLClientKeystoredb` and `SSLCClientKeystash` 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 need to access a DB2 for z/OS server system can only do so 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 the use of TLS/SSL. The network is protected by a firewall or other sufficient network security technologies. As a result, in this type of environment, we need to perform the following configuration steps described in the next sections:

1. 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
GSKit key database versus z/OS key rings

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

- Storage of one’s personal certificate and the private key 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 only known to us) is used to sign certain TLS/SSL handshake messages, and to decrypt messages that are encrypted by remote peers 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 (certificate authority certificates and self-signed peer certificates).

Like RACF, GSKit key fulfills both of these purposes using a single repository called a key database. As we will see, configuration of the 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.

Configuring TLS/SSL support in a DB2 Connect server for Linux, UNIX, and Windows

In our scenario, we configure the DB2 Connect server to support TLS/SSL to create a secure connection between the remote client and the server. The connection between the DB2 Connect server and the DB2 for z/OS server can operate in the clear, because it is established in a trusted security zone.

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 GSKit libraries from the IBM DB2 Support Files for SSL Functionality DVD. Alternatively, you can install the GSKit libraries from an image that you have downloaded from Passport Advantage.

For information about the GSKit tool GSKCapiCmd, see the GSKCapiCmd User’s Guide, which is available on the following web page:


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 GSKit libraries is included in the PATH environment variable. For example, on Windows, add the GSKit (V7 or V8) bin and lib directories to the PATH environment variable:

  set PATH="C:\Program Files\IBM\gsk7\bin";"C:\Program Files\IBM\gsk7\lib";%PATH%
  or

  set PATH="C:\Program Files\IBM\gsk8\bin";"C:\Program Files\IBM\gsk8\lib";%PATH%
On Linux and UNIX platforms, the GSKit libraries are in `sqlib/lib` or `sqlib/lib64`. So, to set the `LIBPATH`, `SHLIB_PATH` or `LD_LIBRARY_PATH` environment variable, you issue the following commands:

```bash
export LIBPATH=$INSTHOME/sqlib/lib:$LIBPATH:.
export SHLIB_PATH=$INSTHOME/sqlib/lib:$SHLIB_PATH:.
export LD_LIBRARY_PATH=$INSTHOME/sqlib/lib:$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 if 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 need to create a server key database to manage the digital certificates. Next, the DB2 instance owner must configure the DB2 instance for TLS/SSL support:

1. Create a server key database and the associated digital certificates 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 command-line 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\gsk7\bin` (for DB2 V9.5 FP2) or `C:\Program Files\IBM\gsk8\bin` (for DB2 V9.7) on a 32-bit and 64-bit Windows platform. On 64-bit platforms, the 32-bit 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 `sqlib/gskit/bin` on Linux and UNIX platforms.

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

   ```bash
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", issue the following `GSKCapicmd` command:

   ```bash
gsk8capicmd -cert -create -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 of the remote clients that connect to this server using TLS/SSL.

For example, to extract the server self-signed certificate that you created from the prior step, issue the following **GSKCapiCmd** 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 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. Specify, as an example, 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. Specify, as an example, 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.

---

**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.

---

c. Specify, as an example, 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, TCPIP 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 a cipher suite for the server to use for TLS/SSL by setting the SSL_CIPHERSPECS database configuration parameter. We suggest that you leave this SSL database configuration parameter unspecified so that the GSKit can select the strongest available cipher suite that is supported by both the client and server.
GSKit supports the following cipher suites:

- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA

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 and 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, for example:

```
        db2stop
        db2start
```

Configuring Java or non-Java applications to use TLS/SSL to access the DB2 Connect server

To configure Java applications using the IBM DS Driver for JDBC and SQLJ (Type 4 connections) to use TLS/SSL, see “Configure connections under the IBM DS Driver for JDBC and SQLJ to use TLS/SSL” on page 44.

To configure non-Java DB2 clients for Linux, UNIX, and Windows to use TLS/SSL, see “Configuring the IBM DS Driver non-Java interfaces, such as CLI/ODBC and Microsoft .NET” on page 48.

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

Client access to DB2 using TLS/SSL client authentication

With DB2 for z/OS, a remote client application can access a DB2 for z/OS server using the TLS/SSL client authentication model (which is described in “Coding the AT-TLS policy rules for TLS/SSL client authentication” on page 23 and “Register a client certificate with RACF (optional)” on page 35).

**Note:** To enable this support, authorized program analysis reports (APARs) PM37057 and PM53450 need to be installed on the DB2 10 server.

The DB2 for z/OS server can perform advanced client authentication checking, which builds upon the basic TLS/SSL client authentication model. With this support in place, the DB2 for z/OS server uses the z/OS user ID associated with the client's digital certificate, in addition to that 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 using digital certificates, we briefly explore the DB2 for z/OS processing to authenticate a remote client accessing DB2 using digital certificates.
Processing client access 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 11.

The following numbers correspond to the steps in Figure 11:

1. If the connection is accepted to a secure port, TLS/SSL protection of the connection must be verified. In order for DB2 to determine if the connection is secure, DB2 issues the AT-TLS SIOCTTLSCTL input/output control (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 user ID 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 currently enabled for the connection), the connection is not secure.

   If the connection is not secure and the remote connection is trying to access DB2 via secure port, DB2 rejects the connection request. Otherwise, DB2 enables the connection, at least long enough to perform the following additional steps.

   Step 1: Verify secure nature of connection?
   - Non-secure connection trying to access DB2 via secure port, reject request

   Step 2: Is authentication information present?
   - TCPALVER=NO
   - TCPALVER=YES or connection is secure and security type indicates Server with client authentication, ClientAuthType = SAFCheck

   Step 3: Does the serving subsystem accept remote requests without verification?
   - Y
   - N

   Step 4: Check ID for connections
   - Not authorized; reject request

   Step 5: Verify user identity.
   - Y
   - N

   Step 6: Authenticate the user ID by RACF, verify ID can access DB2, and run connection exit routine (DSN3@ATH).

   Step 7: Check local privilege at the server.

   No trusted context; reject request

   Connection processing

Figure 11  Processing of inbound TCP/IP connection request at a DB2 for z/OS server
2. DB2 checks to see if 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 if DB2 accepts IDs without authentication information:
   - If TCPALVER=NO or TCPALVER=SERVER, DB2 requires the minimum of a user ID and a password.
   - If TCPALVER=SERVER_ENCRYPT, DB2 requires a user ID and a password. In addition, it requires that the security credentials are Advanced Encryption Standard (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 non-encrypted security credentials are accepted only when the connection is secured by the Internet Protocol network.
   - If TCPALVER=YES or TCPALVER=CLIENT, DB2 accepts TCP/IP connection requests that contain only a user ID.
   - If the connection is secure (which was described in step 1 on page 57) and the user ID that is associated with the client certificate is known, DB2 also accepts TCP/IP connection requests that contain only a user ID.

4. DB2 calls RACF to verify the user ID. The user identity (ID) 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 user ID 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 user ID is derived from a distributed name filter that is defined in RACF.

   To use distributed name filters in this situation, DB2 starts RACF’s distributed identity propagation function to retrieve the RACF user ID 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 user ID.

   If RACF is unable to locate a distributed identity name filter for the distributed identity and registry name pair, DB2 treats the distributed identity as a RACF user ID and starts RACF to verify it together with the password, if provided.

5. The user ID is verified and authenticated in the following manner:
   - When Kerberos tickets are used, the RACF user ID is derived from the Kerberos principal identity. To use Kerberos tickets, ensure that you map Kerberos principal names with RACF user IDs.
   - When the remote client accesses DB2 using a secure connection and a RACF user ID is associated with the client certificate, DB2 compares the remote client user ID with the certificate user ID.
   - Assuming that the remote client user ID and the user ID that is associated with the client certificate differ, additional authentication processing by DB2 is required to authenticate the remote client accessing DB2 using a certificate.
– If the remote client user ID and the user ID that is associated with the client certificate are the same, DB2 authenticates the user ID (and password, if provided) with RACF, normally.

**Important:** Additional authentication processing is required to authenticate the remote client accessing DB2 using a certificate, as explained next.

Additional authentication is required to authenticate the remote client connection when the remote connection request accesses DB2 using a digital certificate, and the remote client user ID differs from the RACF user ID that is associated with the client certificate. DB2 performs additional steps:

– The user ID that is associated with the client certificate is authenticated by RACF. This user ID is also verified by RACF to ensure that it can access DB2 and the connection exit routine is started.

– Assuming that the authentication is successful, DB2 searches for a matching trusted context that is defined for the user ID 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 user ID that is associated with the remote connection request. This user ID is also verified by RACF to ensure that it can access DB2 and the connection exit routine is started.

Assuming that the authentication is successful, DB2 determines if the primary user ID is enabled to use the trusted connection. If the trusted connection can be used, the remote client is allowed access into DB2 using a digital certificate.

If authentication is unsuccessful for the remote client user ID, or the remote client user ID is not allowed to use the trusted connection, the remote connection request is rejected.

– If authentication is unsuccessful for the client certificate user ID, or a trusted connection was not able to be established for the client certificate user ID, the remote connection is rejected.

If special consideration to authenticate the remote client connection request is not required, the user ID together with the password or RACF PassTicket, if provided, is authenticated by RACF. DB2 calls RACF to check the user ID’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 of the following conditions are true:

– The current operating system is z/OS V1.5 or later.
– The Internet Protocol 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 Internet Protocol network access control definitions in the TCP/IP profile. If one or more of these conditions are not true, RACF uses the literal string TCP/IP.

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 translated.) 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 user ID on a transaction boundary. A sequence of tasks is then performed by DB2 when the remote connection requests to switch the user ID on a trusted connection:

- DB2 determines if 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 is to establish a valid authorization ID to be associated with the trusted connection. Until a valid authorization is established, if any SQL statement is issued, an error (SQLCODE -900) is returned.

Configuring client access to DB2 using TLS/SSL client authentication

To implement client access to a DB2 for z/OS server using digital certificates, you need to perform the following steps:

- Configure the AT-TLS policy rules for the DB2 server to use TLS/SSL client authentication (which was described in “Coding the AT-TLS policy rules for TLS/SSL client authentication” on page 23 and “Register a client certificate with RACF (optional)” on page 35).

- Optionally, create a trusted context definition for the client certificate RACF user ID as the system authorization ID and allow any DRDA user ID to be used by the trusted context.

Example 26 shows how to create a trusted context for the user ID, USRT001, which is associated with the registered client certificate, and allow user IDs USRT020, USRT021, and ADMF002 to use this trusted context.

```
Example 26 Sample trusted context definition for client access to DB2 using certificates

CREATE TRUSTED CONTEXT CTX1
BASED UPON CONNECTION USING SYSTEM AUTHID USRT001
ATTRIBUTES (ADDRESS '9.30.223.20')
NO DEFAULT ROLE
ENABLE
WITH USE FOR USRT020,USRT021,ADMF002;
```

DB2 is now ready to accept remote client access using a digital certificate that is registered to RACF, which has a user ID, USRT001, that is associated with it.

In the case of Example 26, where a connection is established from an application server originating from IP address 9.30.223.20 authenticated with the certificate associated with user USRT001, the connection can switch the DB2 primary authorization ID to either user IDs USRT020, USRT021, or ADMF002 without requiring any credentials.

User ID: With client access to DB2 using a digital certificate, the user ID that the client application specifies for the connection must also be a RACF-defined user ID. It does not have to be the same user ID as the certificate user ID, but the user ID must be known to RACF, and have access to DB2 resources (DSNR class).

In the case where the user ID that is specified by the remote client application is a distributed user ID, which is registered to a distributed registry name (the user ID 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 user ID and one or more distributed user identities. You can use the RACF RACMAP command to associate a distributed user identity with a RACF user ID. RACF distributed identity filters are implemented through z/OS identity propagation. Distributed identity filters are supported on z/OS V1R11 and later releases.

DB2 provides support for z/OS identity propagation and distributed identity filters. You need to create distributed identity filters to take advantage of this support.
Follow these steps to create a distributed identity filter:

1. Activate the RACF general resource IDIDMAP class and enable it for RACLST processing by issuing the following command:
   
   ```
   SETROPTS CLASSACT(IDIDMAP) RACLIST(IDIDMAP)
   ```

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

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

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

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

4. If necessary, review the distributed identity filter by issuing the following RACMAP LISTMAP 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 user ID, USRT021, when the distributed identity is user MARY from Registry01. When user MARY authenticates her identity at her distributed application server and performs tasks that access a remote DB2 server system, DB2 passes distributed user name 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 user ID. Because the remote client accesses DB2 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 user ID. If in place, audit records for this transaction contain RACF user ID USRT021, distributed user MARY, and registry name Registry01, which DB2 passes to RACF.
References

For additional information, see the following publications:

- Security Functions of IBM DB2 10 for z/OS, SG24-7959
- 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 V2R1 Communications Server TCP/IP Implementation Volume 2: Standard Applications, SG24-8097
- IBM z/OS V2R1 Communications Server TCP/IP Implementation Volume 4: Security and Policy-Based Networking, SG24-8099
- DB2 10 for z/OS Administration Guide, SC19-2968-04
- z/OS V2R1 Communications Server: IP Configuration Guide, SC27-3650
- z/OS V2R1 Communications Server: IP Configuration Reference, SC27-3651
- z/OS Communications Server IP Diagnosis Guide, GC27-3652
- z/OS V2R1 Communications Server: IP Messages Volume 2 (EZB, EZD), SC27-3655
- z/OS V2R1 Security Server RACF Command Language Reference, SA23-2292
- z/OS V2R1 Security Server RACF Security Administrator’s Guide, SA23-2289
- z/OS V2R1 Cryptographic Services System Secure Sockets Layer Programming, SC14-7495
- IBM DB2 9.7 for Linux, UNIX, and Windows Developing Java Applications, SC27-2446
- GSKIT return codes:

Authors

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

Chris Meyer is an IBM Senior Software Engineer on the z/OS Communications Server design team, focusing on communications security. He has 30 years of experience developing IBM operating systems and security-related software products.

Paolo Bruni is an International Technical Support Organization (ITSO) Project Leader based in IBM Silicon Valley Laboratory. Paolo has authored several IBM Redbooks® publications and IBM Redpapers™ documents about DB2 for z/OS.
Thanks to the following people for their contributions to this project:

Gayathiri Chandran
He-Joon Kim
Yong Kwon
Jim Pickel
IBM Silicon Valley Laboratory, San Jose

Andy Tracy
IBM Research Triangle Park

Mary Ann Ritosa
IBM Poughkeepsie

Sean A Dunn
DB2 Consultant, Washington D.C.

Thanks to the authors of the previous edition of this paper, *DB2 for z/OS: Configuring TLS/SSL for Secure Client/Server Communications*, published in October 2011:

Derek Tempongko
Paolo Bruni

**Now you can become a published author, too!**

Here is an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Obtain more information about the residency program, browse the residency index, and apply online on the following web page:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

**Stay connected to IBM Redbooks**

- Find us on Facebook:
  [http://www.facebook.com/IBMRedbooks](http://www.facebook.com/IBMRedbooks)
- Follow us on Twitter:
  [http://twitter.com/ibmredbooks](http://twitter.com/ibmredbooks)
- Look for us on LinkedIn:
  [http://www.linkedin.com/groups?home=&gid=2130806](http://www.linkedin.com/groups?home=&gid=2130806)
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
- Stay current on recent Redbooks publications with RSS Feeds:
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features described in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
Send us your comments in one of the following ways:

- Use the online Contact us review Redbooks form:
  ibm.com/redbooks
- Send your comments in an email:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD  Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the web:


The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

DB2®
DB2 Connect™
Distributed Relational Database Architecture™
DRDA®

IBM®
MVS™
Passport Advantage®
RACF®
Redbooks®

Redpaper™
Redpapers™
Redbooks (logo) ®
z/OS®

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.