Protegrity z/OS Protector Guide 7.0.1

Created on: Aug 8, 2024

Notice

Copyright

Table of Contents

Chapter 1

Introduction To This Guide

This guide provides information about installing, configuring, and using the Protegrity z/OS Protectors.

1.1 Sections contained in this Guide

The guide is broadly divided into the following sections.

- Section *1 Introduction to this Guide* defines the purpose and scope for this guide. In addition, it explains how information is organized in this guide.
- Section *2 Overview of the Protegrity z/OS Protectors* provides an overview of the Protegrity z/OS Protectors including the general architecture. In addition, it explains the components, features, and Limitations of the z/OS Protector.
- Section *3 Prerequisites for z/OS Product Installation* contains the prerequisites for the installation of the z/OS Protectors.
- Section *4 Installing the z/OS Product* demonstrates the installation instructions for the z/OS Protectors. In addition, this section describes how to upgrade the z/OS Protector.
- Section *5 Application Protector on Mainframe z/OS* describes the Application Protector variants on z/OS, including installation and configuration for both stateful and stateless protocols.
- Section *6 File Protector Solutions on Mainframe z/OS* describes the deployment and management of permissions related to File Protector on z/OS platform.
- Section *7 Database Solutions on Mainframe z/OS* discusses about the Database Solutions on z/OS Protectors. In addition, it describes how to install and configure the Mainframe z/OS Protector. A list of all the UDFs with the syntax are provided.
- Section *8 IMS Solutions on Mainframe z/OS* describes about IMS Protector for z/OS including the installation and configuration. In addition, this section describes about the IMS segment edit/compression exit routines including restrictions and limitations, usage of compression and decompression, return and reason codes, and sample programs.
- Section *9 Appendix A: Uploading Certificates from ESA to z/OS Manually* demonstrates how to upload certificates from ESA to z/OS manually.
- Section *10 Appendix B: Sample Codes for Automated Installation* contains sample codes for Automated Installation.
- Section *11 Appendix C: Sample Code for PTYPARM Configuration file* demonstrates the usage of the PTYPARM configuration file that is used for customization of Cryptographic Servers on the Mainframe z/OS platform.
- Section *12 Appendix D: Sample RACF Setup for the Started tasks* contains sample code to setup for the RACF permissions in the Protegrity and PEP server started tasks.
- Section *13 Appendix E: z/OS Application Protector Samples* contains samples of Application Protector for Mainframe.
- Section *14 Appendix F: Sample Code for the z/OS Database Protector Mainframe* contains samples of Database Protector for Mainframe.
- Section *15 Appendix G: Sample Code for File Protector Mainframe* contains sample code to perform various tasks with File Protector for Mainframe.

- Section *16 Appendix H: Sample Code for IMS Protector Mainframe* contains sample codes for performing various tasks with IMS Protector for Mainframe.
- Section *17 Appendix I: Security Considerations* contains suggestions about how to use the external security manager of z/OS to provide additional security for the Protegrity z/OS protector software.

1.2 Accessing the Protegrity documentation suite

This section describes the methods to access the *Protegrity Documentation Suite* using the *My.Protegrity* portal.

1.2.1 Viewing product documentation

The **Product Documentation** section under **Resources** is a repository for Protegrity product documentation. The documentation for the latest product release is displayed first. The documentation is available in the HTML format and can be viewed using your browser. You can also view and download the *.pdf* files of the required product documentation.

1. Log in to the *My.Protegrity* portal.
2. Click **Resources** > **Product Documentation**.
3. Click a product version.
   The documentation appears.

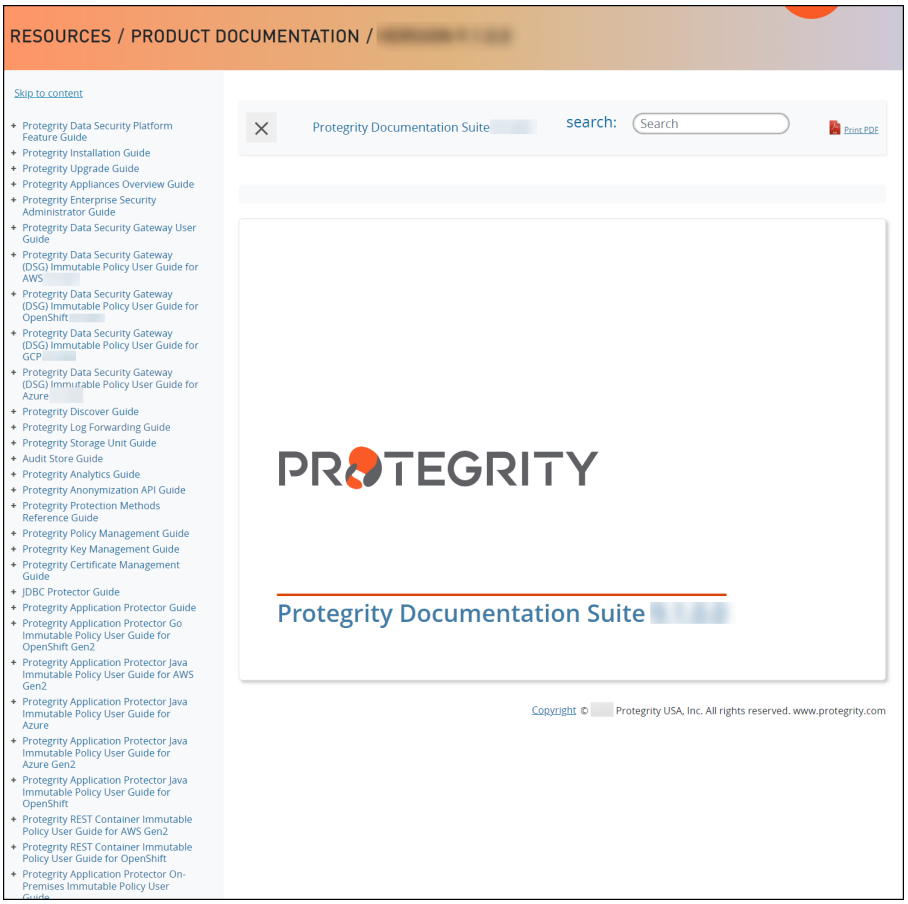

Figure 1-1: Documentation

4. Expand and click the link for the required documentation.
5. If required, then enter text in the **Search** field to search for keywords in the documentation.
   The search is dynamic, and filters results while you type the text.
6. Click the **Print PDF** icon from the upper-right corner of the page.

The page with links for viewing and downloading the guides appears. You can view and print the guides that you require.

1.2.2 Downloading product documentation

This section explains the procedure to download the product documentation from the *My.Protegrity* portal.

1. Click **Product Management** > **Explore Products**.
2. Select **Product Documentation**.
   The **Explore Products** page is displayed. You can view the product documentation of various Protegrity products as per their releases, containing an overview and other guidelines to use these products at ease.
3. Click **View Products** to advance to the product listing screen.
4. Click the **View** icon ( ) from the **Action** column for the row marked **On-Prem** in the **Target Platform Details** column.
   If you want to filter the list, then use the filters for: **OS**, **Target Platform**, and **Search** fields.
5. Click the icon for the action that you want to perform.

Chapter 2

Overview of the Protegrity z/OS Protectors

This section provides an overview of the Protegrity z/OS Protectors that includes the general architecture, components, features, and limitations of the z/OS Protector.

The Protegrity Data Security Platform is comprised of the following components:

- **Enterprise Security Administrator (ESA)** – The ESA serves as a single point of management for creating and deploying data security policies, rules configuration, and monitoring of the system on an on-going basis.
- **Data Protectors** - The Data Protectors include the Application Protector, Database Protector, File Protector, and IMS Protector. The protectors are used to protect sensitive data in the enterprise and deploy security policy for enforcement on each installed system.

The ESA working in combination with a data protector can be used to encrypt or tokenize sensitive data. The PEP server provides a platform server that interfaces between the Protegrity protectors and the ESA.

The Protegrity z/OS Cryptographic Services Module (CSM) is a highly secured component that provides data protection and unprotection to the following data protectors: Application Protector, File Protector, Database Protector, and IMS Protector. The Protegrity z/OS protectors can protect DB2 and IMS databases, VSAM data, and flat file data. The Cryptographic Services Manager is not required for User Defined Functions (UDFs).

2.1 Architecture

The Protegrity z/OS Protector architecture is explained in this section.

The PEP server runs under Open MVS (OMVS) and communicates with the ESA via encrypted TCP/IP communications. It puts the data protection policy into UNIX Interprocess Communications (IPC) shared memory segments. These segments are shared with the Cryptographic Services Module. The CSM then provides services to the various protectors through cross memory calls.

The z/OS specific protectors are basically functions that build a parameter block from application input, and then call CSM for policy enforcement and data protection or unprotection. The following figure shows the Protegrity z/OS Protector architecture.

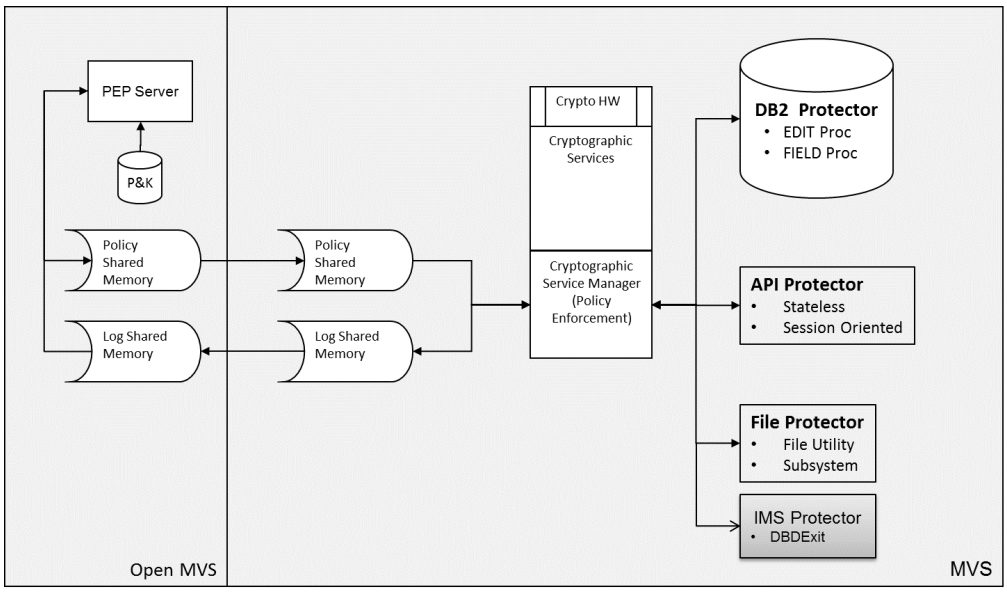

Figure 2-1: Protegrity z/OS Protector Architecture

2.2 Components of the z/OS Protector

This section explains all the components of the z/OS Protector.

The following are the components of the z/OS Protector:

- *Protection Enforcement Point Server*
- *Cryptographic Services Module*
- *DB2 Database Protectors*
- *Application Protectors*
- *File protectors*
- *IMS Protectors*

2.2.1 Protection Enforcement Point (PEP) Server

The PEP server, one of the key components of the z/OS Protector, is discussed in this section.

The PEP server is an UNIX compatible cross-platform server that loads the data security policy from the ESA, and makes it available for the various protectors. It also provides audit log services to send audit information back to the ESA. It is written in C and C++ and thus requires the IBM Language Environment to run. Protegrity provides two sample Job Control Language (JCL) started tasks to start the PEP server and to stop it from MVS. These are designed to be used by System Automation.

2.2.2 Cryptographic Services Module

The Protegrity z/OS Protectors include an additional cryptographic server. The Cryptographic Service Manager (CSM) handles management of protecting and unprotecting data through a cross-memory interface.

The Cryptographic Service Manager (CSM) allows centralized processing of the data and protects the encryption keys and token tables from rogue user programs.The CSM is used by the Application Protector, Database Protector, File Protector, and IMS Protector PEPs (FIELDPROC, EDITPROC, Cryptographic Service Interfaces, File Utility, and CSS) on Mainframe z/OS, but is not used by the DB2 User Defined Functions.

2.2.3 DB2 Database Protectors

The DB2 Database Protectors, one of the components of the z/OS Protector, is discussed in this section.

Protegrity has three ways to protect data in a DB2 database.

- FIELDPROC
- EDITPROC
- UDFs

The DB2 FIELD PROCEDURE exit runs as part of the xxxxDBM1 task of the DB2 region. The FIELDPROC is defined in the DDL as part of the column definition.

The EDIT PROCEDURE exit runs as part of the xxxxDBM1 task of the DB2 region. This exit protects the ROW, and not column data. EDITPROC is also defined in the DB2 DDL, but is defined at the TABLE level and not as part of the COLUMN definition. Since DB2 restricts EDITPROC to no parameters, Protegrity has a customization process to add the data protection method to the EDITPROC program.

The User Defined Functions (UDFs) are available for use with VIEWs and TRIGGERs, but not all the UDFs are available on z/OS due to DB2 limits on UDF on z/OS. The UDFs allow the most application control for data protection and unprotection, but are also much more complicated to implement. UDFs are usually used in conjunction with VIEWs and TRIGGERs.

2.2.4 Application Protectors

Protegrity has multiple application interfaces to protect data in applications including VSAM and IMS data. These interfaces include Language Environment (LE) compatible and non-compatible interfaces. The interfaces are either session oriented or stateless. The two types have the same parameter lists within the type for the LE version and the non-LE version. The stateless ones have been tested with CICS of various levels.

The APIs protect or unprotect data based upon the algorithm specified by the data element used. The interfaces only protect or unprotect the data within the buffer passed by the application and do not care where the data comes from (file or working storage). The interfaces are standard z/OS functions and have been used with COBOL, PL/I and ADABAS Natural applications.

2.2.5 File Protectors

An overview of the File Protectors, one of the components of the z/OS Protector, is discussed in this section.

Protegrity has two ways to protect data in flat files (QSAM files).

- Cryptographic File Utility (CFU)
- Cryptographic Sub System (CSS)

Both methods work with generation data groups as well as normal files.

The CFU is a file protection utility like IBM’s IEBGENER. This method has two input files and one output file. One input file contains the data to be protected or unprotected, and the other input file contains the control information to tell the program what protection method, what data columns and what records are to be protected. The output file contains the protected or unprotected data that was processed. Only one set of columns or records can be processed at a time. The entire file is processed at once.

The CSS has two flavors – *PTYPAMLT* and *PTYPAMEX*. Both use the JCL DD statement SUBSYS capability to protect or unprotect data in the input or output buffer. One method keeps the same record format, as the output data. This method only supports length preserving algorithms. The other method adds header records to the output data and uses that information to reformat the data on input as the program expects.

2.2.6 IMS Protectors

A general overview of the IMS Protectors is discussed in this section.

Protegrity has three ways to protect data in the IMS database.

- PTYPIMS DBDEXIT

- PTYPIMSC DBDEXIT
- Protegrity z/OS Application protector (*PTYPSLL*/*PTYPSLI*)

The IMS Segment edit/compression exit routine, *PTYPIMS* will protect the data in IMS database during LOAD/INSERT and unprotect the data during SELECT.

The IMS Segment edit/compression exit routine, *PTYPIMSC* will compress the repetitive data in IMS database before protect during LOAD/INSERT and after unprotect the data during SELECT, it will decompress the data.

The data in the IMS database can also be protected/unprotected using the z/OS Stateless Application protectors – *PTYPSLL* and *PTYPSLI*.

2.3 Features of z/OS Protector

The features of z/OS Protector are discussed in this section.

All the z/OS specific protectors are written in mainframe assembler and are designed explicitly to run on z/OS. They are re-entrant, re-usable, and all run in 31-bit mode or 64-bit mode as needed. Where necessary (such as the APIs), the programs are Language Environment compatible.

All data protection and unprotection plus the enforcement of the Protegrity data security policy is done away from the user address space in the protected CSM address space. All keys and other information needed for data protection are in fetch protected storage to restrict access.

All data protection and unprotection is done on z/OS.

2.4 Limitations

This section lists the limitations of the z/OS Protectors.

- The z/OS protectors do not support all the token data types that are supported on other platforms.
- The z/OS UDFs do not support data types like LONG VARCHAR and BLOB.
- The z/OS Protectors (except z/OS UDFs) do not support Null handling.
- The z/OS Protectors do not support the DTP2 algorithms.

2.5 Supported Database Matrix

This section lists the supported database matrix for the z/OS Protectors.

Table 2-1: Supported Database Matrix

Database	Supported Protectors	Supported Platforms
DB2 on z/OS	DB2 9.1	z/OS 2.1
	DB2 10.1	z/OS 2.2
	DB2 11.1	z/OS 2.3
	DB2 12.1	z/OS 2.4
	FIELDPROC	z/OS 2.5
	EDITPROC	

Database	Supported Protectors	Supported Platforms
	UDFs	
IMS on z/OS	v13 v14 v15	z/OS 2.1 z/OS 2.2 z/OS 2.3 z/OS 2.4 z/OS 2.5

Chapter 3

Prerequisites for Installing the z/OS Product

This section describes the z/OS system requirements for product installation.

3.1 PEP Server on Open MVS (Unix System Services)

This section provides information about the prerequisites and additional setup requirements for installing the PEP server on Open MVS.

3.1.1 Environment Requirements

This section describes about the environment requirements for the z/OS protector installation.

Before setting up the PEP server for Open MVS (UNIX System Services), ensure that your environment meets the minimum requirements. The following table shows the environment requirements.

Table 3-1: Environment Requirements

Free Disk Space	Configuration
100 MB per node	PEP server and repository
262 MB (67072 pages)	Inter-process communications memory. This can be broken down to 200 MB for tokens, 2 MB for policy, 2 MB for policy information, and 8 MB for logs. This is used for communications between the various servers. If tokens are not going to be used, you need to modify the `pepserver.cfg` file to disable tokenproc. This reduces the size by 200 MB.

Note: If the Log Server is not going to be continuously active, then you will need additional storage for the PEP server ZFS file system to maintain audit log records until the Log Server is active.

You can run the MVS operator command `d omvs,o` to view current configuration settings. Review the `IPCSHMMPAGES` value representing the number of 4K pages that are available.

Run the `set IPCSHMMPAGES` command to increase the value, if required. The *BPXPRMxx* member used for system IPL must be updated to set the increased value to permanent.

Run the `SETOMVS IPCSHMMPAGES` command to increase the number of shared memory pages.

3.1.2 Setup Requirements

This section contains the additional setup requirements that should be met before you can proceed with the PEP server installation

To setup:

1. It is strongly suggested that a separate mount point should be used for installing the PEP server. This mount point can be wherever you want, with the default and documented mount point of `/opt/protegrity`. The underlying file can be HFS or ZFS and needs to be at least 100 MB in size with expansion possible, and 200 MB in size if tokens are used. The PTY701Z2 sample JCL in `ptyhlq.SPTYSAMP` can be modified and used to create a ZFS mount point for the PEP server.

   > **Note:** The HFS files can be created from the ISPF 3.2 menu and ZFS files can be created from the tso ishell (Refer to the cryptographic server installation information below for who needs what access rights to the directories that will be created here). The `.../defiance_dps/bin` subdirectory can be shared across LPARs that will run the same version of the PEP server, but the `.../defiance_dps/data` subdirectory needs to be unique for each LPAR. The $SYSNAME system variable can be used in the various scripts and JCL to make the subdirectory unique.

2. Ensure that the user starting the PEP server, has `BPX.DAEMON` rights.

   > **Note:** The installing user needs External Security Manager (such as RACF) and Open MVS access rights to do the install.

3. Ensure that the PEP server port listening for the policy changes corresponds to the port that the ESA will use to deploy the policy to, and is open in your Firewall setup for in-bound traffic.
4. Ensure that the IP address and the port number that the Log Server will be listening on to receive audit logs, is open in your Firewall setup for out-bound traffic.
5. For the PEP server user, set the following parameters in either the installation's `/etc/profile` or `/HOME/.profile`:
   - _BPX_SHAREAS=YES
   - _BPX_SETIBMOPT_TRANSPORT=TCPIP
6. Ensure that the localhost is correct.

   For z/OS, run the following command in Open MVS command prompt: **`host localhost`**

   The localhost IP address should be 127.0.0.1 or the system's IP address. If you receive an error message instead, such as `Hostname lookup failure` or `Unknown`, then perform the following steps:

   a. Remove the IP address value from the DNS.

   b. Configure the *TCPIP.DATA* to specify the *LOOKUP LOCAL DNS*.

   This ensures that `/etc/hosts` is checked prior to the DNS.

   > **Note:**
   > If the host details are present in some customized file, instead of the default locations like `etc/hosts`, `DNS resolution`, or `TCPIP.DATA`, then perform any of the two following suggestions to resolve hostname:
   > - Use the *ENVAR* script to set the *RESOLVER_CONFIG* variable that points to the file in the JCLs executing the TCP/IP functions.
   > - Add the *RESOLVER_CONFIG=//'<dataset-name>'* setting in the *PTY$ENV* file that is present in the *SPTYSAMP* file.

3.2 Cryptographic Servers on z/OS MVS

This section provides information about setting up the Cryptographic Servers on z/OS MVS.

3.2.1 Setup Overview

This section describes about setting up the Cryptographic Servers on the z/OS MVS.

Before starting the installation, you should check that you have the following items available and ready:

- Servers
- Distribution Files
- Modules

3.2.1.1 Servers

In addition to the Protegrity z/OS protector, you need a workstation(s) running the ESA which manages data security policy, key management, and auditing and reporting.

3.2.1.2 Distribution Files

The products are distributed in three files (*SPTYAUTH*, *SPTYAUTX*, and *SPTYSAMP*) and a *ZFS.DPSxxxx.INSTALL* file.

The following table describes the Distribution Files.

Table 3-2: Distribution Files

File	Description
SPTYAUTH	Contains the necessary executable modules.
SPTYAUTX	Contains executables to interface with the PEP server under UNIX System Services.
SPTYSAMP	Contains the sample JCL.
ZFS.DPSxxxx.INSTALL	Contains the PEP server and the necessary support files.

3.2.1.3 Modules

The Protegrity z/OS Protectors consists of three modules.

- The Cryptographic Services Manager (*PTYPCSM* module and its associated modules)
- The PEP server executable and its pieces, which runs under Open MVS (UNIX System Services)
- The actual data protectors

The following table describes these modules.

Table 3-3: Protegrity z/OS Protector Modules

Module	Description
Cryptographic Services Manager (*PTYPCSM*)	The *PTYCSRV* procedure starts the *PTYPCSM* module and its associated modules. This task needs to be APF authorized and provides the basic encryption services including the security policy enforcement. The task consists of management pieces and various service provider sub-tasks. It provides a command interface to allow monitoring and some dynamic reconfiguration. During startup, the main task reads a parameter file to get its startup information. Use the MVS stop command to close this server. You can also optionally use this command to stop *PTYSPEPS*.
PEP server	The *PTYSPEPS* starts the Open MVS daemon that provides the policy and handles the audit log records. *PTYPPEPS* stops the daemon.

A sample code to start these servers is included with the product. The appropriate External Security Manager, such as RACF and Top Secret needs to be set up to allow these started tasks to execute with the appropriate authorities.

3.2.1.4 Cryptographic Hardware

The IBM mainframe hardware (z-architecture) has feature code 3863 which adds cryptographic hardware to the system. This hardware is export restricted but is generally available in the United States of America and other nations.

The default installation for Protegrity z/OS Protector will utilize this hardware if available. If the hardware is not available, then the Database Protectors will do software encryption and decryption. This is significantly slower than the use of the cryptographic hardware.

Contact your IBM representative if you are not sure that the hardware has been turned on. The availability of the hardware feature is decided upon each invocation so that enabling the feature later does not require reinstalling the Protegrity software.

Chapter 4

Installing the z/OS Product

This section explains the installation of the z/OS PEP server on the Open MVS, the Cryptographic Server and the protectors on the MVS.

Before you begin
The Protegrity z/OS product package contains the following three components:

- `.TRS` file - A tersed DFSMS dump of Protegrity version 7.0.1.n MVS files pertaining to the protector and the `SPTYxxxx` files, and a ZFS volume that includes the PEP server shell script and a sample input for the *BPXBATCH* job to install the PEP server.
- `LODxxxx.JCL` – A text file containing the sample JCL to unterse the `.TRS` files.
- `productname_zOS-V2-ALL_Mainframe-ALL_ALL_n.n.n.n.readme` – A readme file briefing on the product.

To install the z/OS product:

1. Upload the tersed files (`.TRS`) to the z/OS system in binary mode. The files should be RECFM FB and LRECL 1024. BLKSIZE= 27648 is any multiple of 1024. In the product zip file, `productname_zOS-V2-ALL_Mainframe-ALL_ALL_n.n.n.n.readme` is a readme file that contains information about changes and suggestions to FTP the files to z/OS.
   ```
   QUOTE SITE RECFM=FB LRECL=1024 BLKSIZE=27648 CYL PRI=150 SEC=20
   ```
2. Upload the `LODxxxx.jcl` to the z/OS system. Customize the JCL to specify the receiving volume in SET VOL statement. The MVS datasets for respective protector and the `ZFS.DPSxxxx.INSTALL` are restored after running this JCL.
   a. The `LODxxxD.jcl` for the Database Protector, files restored in the given VOL are:
      - `PTYxxxx.SPTYAUTH`
      - `PTYxxxx.SPTYAUTX`
      - `PTYxxxx.SPTYSAMP`
      - `PTYxxxx.DB2AUTH`
      - `PTYxxxx.DB2LOAD`
      - `PTYxxxx.DB2SAMP`
      - `PTYxxxx.UDFSAMP`

- *PTYxxxx.UDFSQL*
- *PTYxxxx.UDFUDFX*
- *ZFS.DPSxxxx.INSTALL*

b. The *LODxxxA.jcl* for the Application Protector, files restored in the given VOL are:

- *PTYxxxx.SPTYAUTH*
- *PTYxxxx.SPTYAUTX*
- *PTYxxxx.SPTYSAMP*
- *PTYxxxx.APLOAD*
- *PTYxxxx.APSAMP*
- *PTYxxxx.APSCNTL*
- *PTYxxxx.APSLOD*
- *ZFS.DPSxxxx.INSTALL*

c. The *LODxxxF.jcl* for the File Protector, files restored in the given VOL are:

- *PTYxxxx.SPTYAUTH*
- *PTYxxxx.SPTYAUTX*
- *PTYxxxx.SPTYSAMP*
- *PTYxxxx.FPAUTH*
- *PTYxxxx.FPLOAD*
- *PTYxxxx.FPSAMP*
- *ZFS.DPSxxxx.INSTALL*

d. The *LODxxxF.jcl* for the IMS Protector, files restored in the given VOL are:

- *PTYxxxx.SPTYAUTH*
- *PTYxxxx.SPTYAUTX*
- *PTYxxxx.SPTYSAMP*
- *PTYxxxx.IMSAUTH*
- *PTYxxxx.IMSLOAD*
- *PTYxxxx.IMSSAMP*
- *ZFS.DPSxxxx.INSTALL*

Note:

1. The *ZFS.DPSxxxx.INSTALL* is used during the PEP server Installation, explained in the section *Installing the PEP Server* .
2. The *PTYxxxx.SPTY** libraries are used to install the Cryptographic Server, explained in the section *Installing the Cryptographic Server* .
3. The *PTYxxxx.SPTYSAMP* library contains the sample JCL's to install the Protegrity z/OS product.
4. The *PTYxxxx.AP** libraries are used to install the Application Protectors, explained in the section *Installing and Customizing the Application Protector on z/OS*.
5. The *PTYxxxx.FP** libraries are used to install the File Protectors, explained in the section *Installing and Customizing the File Protector on z/OS*.
6. The *PTYxxxx.DB2** libraries are used to install the Database Protectors (FIELDPROC and EDITPROC), explained in section *Setting Up Database Protector on Mainframe z/OS*.
7. The *PTYxxxx.UDF** libraries are used to install the Database Protectors (UDFs), explained in the section *Installing UDFs*.
8. The *PTYxxxx.IM** libraries are used to install the IMS Protectors, explained in the section *Setting Up the IMS Protector*.

4.1 Installing the PEP Server

The *PEPINST.jcl* is the JCL to use the *BPXBATCH* to install the PEP server.

➤ To install the PEP server:

1. Using the ISHELL or OMVS, create a new directory, for example, dps701, under */usr/local/pty* since the *PTY701Z JCL* will mount the *ZFS.DPSxxxx.INSTALL* file at this path.

   > **Note:** If you want to use some different mount point, edit the *PTYxxxx.SPTYSAMP*(PTY701Z) accordingly.

   > **Note:** Once the PEP server is installed, the volume can be unmounted. To unmount the volume, run the following command:

   /usr/sbin/unmount/usr/local/pty/dps701
2. The following will be restored on the mount point, */usr/local/pty/dps701*:
   - *PepServerSetup_zOS_Mainframe_7.0.1.x.sh*
   - *PepServerSetup_zOS_Mainframe_7.0.1.x.tar.Z*
   - *pepserver.install.input*

   > **Note:** The *pepserver.install.input* file is the STDIN to the *BPXBATCH* step for installing the PEP server. You can edit the *pepserver.install.input* file using the ISHELL or OMVS, and specify your install directory as the third line and the ESA address as the fourth line. For more information about installing the PEP server, refer to the section *Sample Code to use BPXBATCH to install PEP Server*.
3. Run the *PTYxxxx.SPTYSAMP*(PEPINST). The PEP server will be installed in the install directory given in the *pepserver.install.input*.
4. Run the ***GetCertificates*** from the *<install dir>/defiance_dps/bin* directory.

   ./GetCertificates –u <admin user>:<admin user password>

   The Certificates from the ESA will be saved in the *<install dir>/defiance_dps/data* directory. The ***GetCertificates*** script uses CURL, an Open Systems utility, to retrieve the certificates from the ESA. Protegrity has compiled this CURL utility that can be used with Open MVS to retrieve the certificates and other information from the ESA. If the version of CURL that Protegrity has supplied, does not work on your system, then follow the manual method of downloading the certificates that is described in the section *Appendix A: Uploading Certificates from ESA to z/OS Manually*.

   > **Note:** The Integrated Cryptographic Services Facility (ICSF) that is used to generate random numbers, should be started before using the GetCertificates script.

   > **Note:** The installation script stores the installation directory path in the ***pepsrvctrl*** executable script. You may plan to move or copy this installation to a different LPAR or to a different directory path on the install system, for example, when you are testing a patch and wish to make the patch permanent. In such cases, you must edit (using oedit) ***pepsrvctrl*** in the *.../defiance_dps/bin* directory. Modify the line starting with *PTY_BASE_DIR=* to point to the new directory structure. The path provided must be the complete directory structure from the root base. Relative directory structure is not permitted.

The following example shows a sample to set the path:

```
000011 # Base directory where Defiance DPS server is installed.
000012 # E.g. /opt/protegrity/defiance_dps
000013 PTY_BASE_DIR=/opt/protegrity/dps701/defiance_dps
```

Note: Do not delete the default directories created during the PEP server installation.

Note:

The support of various special symbols in the password, depends on the codepage selected on the z/OS Protector. The recommended codepage is 1047.

For additional information on usage of special symbols in passwords, refer to the section *Password Policy for All Appliance Services* in the *Protegrity Appliances Overview Guide 9.1.0.4.*

5. Make the necessary changes in the `pepserver.cfg` file located in the `<install dir>/defiance_dps/data` directory. The sample JCL to update the `pepserver.cfg` file from the MVS side, is demonstrated in the section *Sample JCL to execute script to update pepserver.cfg*.

   Note: If you intend to use more than one PEP server, then you need one `pepserver.cfg` file for each server. The `pepserver.cfg` file must be renamed with a unique name, for example `pepserver1.cfg`, `pepserver2.cfg`, and so on. You would also want to specify a different filename for the `pepserver.log` for each PEP server.

6. Verify the access rights. Ensure that the library path variable for your system is correctly set and the appropriate directories exist.

   Note: The PEP server startup task (sample JCL PTYSPEPS group) should own the install directory structure.

4.1.1 Configuring case sensitivity in PEP Server for Policy users

The PEP server fetches users from the ESA and can treat these users as case sensitive, which is the default, or case insensitive. By configuring the case sensitive parameter, you can enable case insensitive policy user names.

Before you begin
Ensure that you are logged in to the protector machine and the protector is installed and configured.

➤ To enable the case insensitive user name support in the PEP server:

1. Navigate to the additional mount point reference `/opt/protegrity/<installdirectory>/defiance_dps/data/` directory of the protector.
2. Open the `pepserver.cfg` file in a text editor.
3. Under the *[member]* section, configure the case-sensitive parameter as follows.

```
# Specifies how policy users are checked against policy.
# yes = (The default) policy users are treated in case sensitive manner.
```

```
# no = policy users are treated in case insensitive manner.
case-sensitive = no
```

Note: The parameter *case-sensitive* is either marked as a comment or set to “yes” by default.

```
#case-sensitive = yes
```

The value of the *case-sensitive* parameter must be uncommented and then set as “no” to enable case insensitive user name support.

```
case-sensitive = no
```

4. Restart the PEP server.

4.2 Installing the Cryptographic Server

The MVS datasets restored from the *LODxxxx.jcl* will install the Cryptographic server.

To install the Cryptographic Server:

1. The libraries *PTYxxxx.SPTYAUTH* and *PTYxxxx.SPTYAUTX* must be APF authorized or the load modules included must be copied to authorized libraries, and the sample invocation JCLs from *PTYxxxx.SPTYSAMP*, modified to point to the new libraries. The *SPTYAUTX* library must be a PDSE since it contains C LE modules and the interface to call them.
2. Copy *PTYCSRV*, *PTYSPEPS*, and *PTYPPEPS* sample procedures from *PTYxxxx.SPTYSAMP* to your procedure library and customize. This can be done using *PTYxxxx.SPTYSAMP* (JMVS) which will update COMMID and additional arguments in respective procedure and copy the started tasks in procedure library, *USER.PROCLIB*. For more information refer to the section *Details on JMVS jcl to install Cryptographic Server*.

   The required accesses and authorities are:

   - The *PTYCSRV* is the main driving JCL. Needs read authority to the *SPTYAUTH* and *SPTYAUTX* libraries. In addition, it needs read authority to wherever the *PTYPARM* member from the *SPTYSAMP* library is set up. By default, this task will start *PTYSPEPS* started task during startup and *PTYPPEPS* during shutdown.
   - The *PTYSPEPS* is the secondary task and is also started by itself, during the PTYCSRV startup. This task needs Open MVS authority including read and execute access to the *installdirectory/defiance_dps/bin* directory and read/write/execute access to the *installdirectory/defiance_dps/data* directory. Due to Open MVS rules for *BPXBATCH* usages, this started task user will need ***uid(0)*** (or *BPX.SUPERUSER*) authority and *BPX.DAEMON* authority.
   - The *PTYPPEPS* is the tertiary task that stops *PTYSPEPS*. It needs Open MVS authority including read and execute access to the *installdirectory/defiance_dps/bin* directory. Due to the Open MVS rules for the *BPXBATCH* usages, this started task user will need ***uid(0)*** (*BPX.SUPERUSER*) authority.

Note: If the PEP server is stopped incorrectly, or if it goes down, then the policy that exists in the shared memory is removed. If you terminate the PEP server process, then the PEP server shared memory needs to be removed manually (use the ***ipcrm -m id*** command).

If you need to gracefully shut down the PEP server, then run the following command:

/s PTYPPEPS

When the PEP server is started again, the policy will be recovered from the backup table in the PEP server repository.

Note: Do not purge *PTYSPEPS*. It is recommended to cancel *PTYSPEPS* using the following command:

```
/c PTYSPEPS
```

Note: The *PTYCSRV* procedure invokes the Cryptographic Services Manager Task and auto starts the other tasks. All tasks will need to be started before any invocation of any of the database protectors including even the CREATE TABLE to add EDITPROC or FIELDPROC or the ALTER TABLE to add FIELDPROC.

3. Run `s ptycsrv` from the system console, to start *PTYCSRV* and other Protegrity servers.

Note: For the first invocation of the PEP server, use the *PTYSPEPS* procedure to allow enough time to deploy the policy from the ESA server to the new PEP server.

4. Run `p ptycsrv` from the system console, to end *PTYCSRV* and other Protegrity servers.

Note: If you do not care about any audit log records that may still be in memory, then you can run `p ptycsrv immediate` to stop the servers. This is an immediate stop, and any remaining audit log records are discarded.

Run `f ptycsrv,d log` to find out how many audit log records may still be in memory.

4.2.1 Sample Jobs to Verify the Installation

You can verify your installation by running sample jobs and checking their output for each protector that you have installed.

To start/stop the pepserver and cryptographic Server, refer to the following sample jobs:

- *Starting PEP Server*
- *Stopping PEP Server*
- *Starting Cryptographic Services Task*

4.2.1.1 Starting PEP Server

This section describes about the procedure to start the PEP Server.

The user used for the *PTYSPEPS* and *PTYPPEPS* tasks needs to have OMVS RACF permission *BPX.DAEMON* and to have root `uid=0` within the OMVS. The `uid=0` is necessary because the `pepsrvctrl` script used to run the PEP Server changes directories as part of the startup and shutdown procedure.

```
//*-------------------------------------------------------------------
//* PTYSPEPS - Start the Protegrity Policy Enforcement Point Server
//*
//*   Note:  This procedure is automatically started by the
//*          Protegrity Defiance DPS Data Protector startup
//*          started task.  Please do not start it manually
//*          except as instructed during the initial install
//*          or as instructed by Protegrity support personnel.
//*
//* Customize this procedure to meet your requirements by changing the
//* following parameter on the PROC statement:
//*
//* INSTLDIR='path'  HFS directory where the PEP Server is installed.
//*                  This is the same directory that was specified as
//*                  the installation directory in the PEP Server Setup
//*                  Wizard.
//*
//* Note: UNIX file paths are often long and the PARM in the procedure
//*       contains the specified path as well as some other text.  The
//*       length of the PARM field is limited to 100 bytes.
//*
//* The JCL continuation rules for literals is defined by the following
//* extract from the "IBM z/OS V1R7.0 MVS JCL Reference" manual:
```

```
//*
//* To continue a parameter that is enclosed in apostrophes:
//* 1. Extend the parameter to column 71.  Do not code an apostrophe in
//*    column 71 of a JCL statement that is continued.  The system
//*    interprets the apostrophe in column 71 as the final character in
//*    the statement and ignores the continuation.
//* 2. Code // in columns 1 and 2 of the following statement.
//* 3. Continue the parameter in column 16 of the following statement
//*    even if this splits the parameter.  Trailing blanks or commas
//*    within apostrophes do not indicate a continued statement; the
//*    system treats them as part of the parameter.
//*
//* Example (commented out):
//* ...    PROC INSTLDIR='/directory_1/directory_2/opt/protegrity/defia
//*           nce_dps'
//*
//*-------------------------------------------------------------------
//*
//PTYSPEPS PROC INSTLDIR='/opt/protegrity',PTYHLQ=PTY
//PTYSPEPS EXEC PGM=BPXBATCH,REGION=0M,TIME=NOLIMIT,
//             PARM='sh &INSTLDIR/defiance_dps/bin/pepsrvctrl start'
//STDOUT   DD  PATHDISP=KEEP,PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//             PATHMODE=SIRWXU,
//             PATH='&INSTLDIR/defiance_dps/data/ptyspeps.out'
//STDERR   DD  PATHDISP=KEEP,PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//             PATHMODE=SIRWXU,
//             PATH='&INSTLDIR/defiance_dps/data/ptyspeps.err'
//STDENV   DD  DISP=SHR,DSN=&PTYHLQ..SPTYSAMP(PTY$ENV)
//         PEND
```

4.2.1.2 Stopping PEP Server

The sample code in this section describes about the procedure to stop the PEP Server.

```
//*-------------------------------------------------------------------
//* PTYPPEPS - Stop the Protegrity Policy Enforcement Point Server
//*
//*   Note:  This procedure is automatically started by the
//*          Protegrity Defiance DPS Data Protector started
//*          task when shutting down.  Please do not start it
//*          manually unless the Data Protector started task
//*          (default name: PTYCSRV) has abnormally terminated
//*          without stopping the PEP Server.
//*
//* Customize this procedure to meet your requirements by changing the
//* following parameter on the PROC statement:
//*
//* INSTLDIR='path'  HFS directory where the PEP Server is installed.
//*                  This is the same directory that was specified as
//*                  the installation directory in the PEP Server Setup
//*                  Wizard.
//*
//* Note: UNIX file paths are often long and the PARM in the procedure
//*       contains the specified path as well as some other text.  The
//*       length of the PARM field is limited to 100 bytes.
//*
//* The JCL continuation rules for literals is defined by the following
//* extract from the "IBM z/OS V1R7.0 MVS JCL Reference" manual:
//*
//* To continue a parameter that is enclosed in apostrophes:
//* 1. Extend the parameter to column 71.  Do not code an apostrophe in
//*    column 71 of a JCL statement that is continued.  The system
//*    interprets the apostrophe in column 71 as the final character in
//*    the statement and ignores the continuation.
//* 2. Code // in columns 1 and 2 of the following statement.
//* 3. Continue the parameter in column 16 of the following statement
//*    even if this splits the parameter.  Trailing blanks or commas
//*    within apostrophes do not indicate a continued statement; the
//*    system treats them as part of the parameter.
//*
//* Example (commented out):
//* ...    PROC INSTLDIR='/directory_1/directory_2/opt/protegrity/defia
```

```
//*            nce_dps'
//*
//*-------------------------------------------------------------------
//*
//PTYPPEPS PROC INSTLDIR='/opt/protegrity'
//PTYPPEPS EXEC PGM=BPXBATCH,REGION=0M,
//             PARM='sh &INSTLDIR/defiance_dps/bin/pepsrvctrl stop'
//         PEND
```

This procedure is invoked by *PTYCSRV* during shutdown by default to stop the PEP Server. If you customize this procedure as a different name, then you also need to change the *PTYCSRV* procedure. You also need to make the same changes to *PTYSPEPS* to start the PEP Server.

4.2.1.3 Starting Cryptographic Services Task

The sample code in this section shows how to start the cryptographic services task.

```
//*-------------------------------------------------------------------
//* PTYCSRV  - Start the Protegrity Defiance DPS Data Protector
//*
//* Customize this procedure to meet your requirements by changing the
//* following parameters on the PROC statement:
//*
//* PTYHLQ=PTY         Protegrity Defiance DPS high level qualifier
//*
//* START=AUTO         Startup type: AUTO ! WARM ! COLD
//*
//* COMMID=0           Communication id used by the PEP Server
//*                    This value must match the value specified by the
//*                    communicationid parameter in the PEP Server
//*                    configuration file, pepserver.cfg.
//*                    The default now agrees with the PEP Server
//*                    default
//*
//* PEPSPROC=PTYSPEPS  PEP Server startup procedure
//*
//* PEPPPROC=PTYPPEPS  PEP Server shutdown procedure
//*
//*-------------------------------------------------------------------
//*
//PTYCSRV  PROC PTYHLQ=PTY,START=AUTO,COMMID=0,CSTPROC=,
//             PEPSPROC=PTYSPEPS,PEPPPROC=PTYPPEPS,
//             PARMS=
//PTYCSRV  EXEC PGM=PTYPCSM,TIME=1440,REGION=0M,
//             PARM='START=&START,COMMID=&COMMID,CSTPROC=&CSTPROC,PEPSP
//             ROC=&PEPSPROC,PEPPPROC=&PEPPPROC,&PARMS'
//STEPLIB  DD  DISP=SHR,DSN=&PTYHLQ..SPTYAUTH
//         DD  DISP=SHR,DSN=&PTYHLQ..SPTYAUTX
//SPTYPARM DD  DISP=SHR,DSN=&PTYHLQ..SPTYSAMP(PTYPARM1)
//PTYDUMP  DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//         PEND
```

4.3 Activating Protegrity Data Security Platform

After all the components are installed, you can activate the Protegrity Data Security Platform. Separation of duties, which is a key feature in the Protegrity Data Security Platform, is enforced by having different functions in the organization performing different tasks. The activation steps in this section show that the Protegrity Data Security Platform separates the duties of operating the database from configuring and maintaining the security policy. For information on policy creation and deployment, refer to the section *Creating and Deploying Policies* in the *Policy Management Guide 9.1.0.4*. The Protegrity data security policy uses data elements as the primary method of defining the data protection method to be used. However, just knowing the data protection and unprotection method is not enough information. The Protegrity software also needs to know WHO can cause that data protection or unprotection. Protegrity does this be defining roles that then are assigned privileges for data protection or unprotection. The data users are then assigned to these roles by the Security Officer. The Protegrity z/OS protectors use the RACF USERID or the DB2 Authorized user as the policy user. For the purposes of this document, RACF USERIDs is equivalent to Top Secret USERIDs or ACF/2 USERIDs.

To activate the Protegrity Data Security Platform:

1. Extract the RACF (ESM) USERIDs into a flat file that need to access data to be controlled by the policy. Privileges to the data will be defined for each user (or role) during the policy creation. This file needs to be downloaded to the ESA and defined in the Member Source tab of Policy Management.

   Note: You can, alternatively, create a user and a group file for the Member Source server. For more information about Member Source server, refer to the section *Working with Member Sources* in the *Policy Management Guide 9.1.0.4*). If you create the user and group list manually, then keep in mind that the z/OS User ID or the Group ID are in upper case and the length must not exceed eight characters.

2. Configure data stores and member sources in Policy Management.
3. Create a data security policy in Policy Management.

   Note: If you want to transfer protected data to or from a non-z/OS platform, then select non-CUSP algorithm or tokenization.

4. Deploy the policy to PEP server on z/OS data-store.

The Protegrity Data Security Platform is activated.

4.4 Deploying a Test Policy

This section describes the steps about how to build and deploy a test policy. For more information about policy creation and deployment refer to the section *Creating and Deploying Policies* in the *Policy Management Guide 9.1.0.4*.

Before you begin
You need to build and deploy a simple test policy in Policy Management before you can perform installation verification.

To build and deploy a test policy in ESA:

1. Navigate to **Policy Management** > **Data Elements & Masks** to create the data elements.
2. Navigate to **Policy Management** > **Roles & Member Sources** to define Roles with specific access rights to the data to be protected.

   Note: The username must be in uppercase and length of the username must not exceed eight characters.

   Note: The new behavior of policy users inheriting the permissions from the default role for the data elements is applicable for the z/OS Protector 7.0.1 release from the build number 7.0.1.12 and later builds. For more information about inheriting permissions for users, refer to the section *Inheriting Permissions for Users in Multiple Policies and Roles* in the *Policy Management Guide 9.1.0.4*

3. Navigate to **Policy Management** > **Policies & Trusted Applications** to create Policy.
4. Navigate to **Policy Management** > **Data Stores** to create zOS data store.
5. On z/OS, start the PEP server by starting the *PTYSPEPS* procedure.

   Note: This is a one-time procedure to deploy the first policy to the PEP server. Hereafter, the PEP server will be automatically started and stopped by the Cryptographic Service.

6. Navigate to **Policy Management** > **Policies & Trusted Applications** to deploy the policy to z/OS data-store.
7. On z/OS, stop the PEP server by starting the *PTYPPEPS* procedure.
8. Start the PTYCSRV procedure.

4.5 Verifying the Installation

You can verify that the installation is successful by reviewing the messages that appear during the *PTYCSRV* startup on the system console and in the *PTYCSRV* job log.

The following code displays the messages.

```
PTY001I  Protegrity Data Security Platform V7.0.1.1 startup in
PTY001I  progress.
PTY002I  (C) Copyright 2006-2017 Protegrity Corporation
PTY002I  All Rights Reserved
PTY050I  Protegrity Data Security Platform startup parameters:
PTY050I  START:          AUTO
PTY050I  COMMID:                          0
PTY050I  CSTPROC:
PTY050I  PEPSPROC:       PTYSPEPS
PTY050I  PEPPPROC:       PTYPPEPS
PTY050I  LOGQBS:                         10
PTY050I  LOGMQB:                        123
PTY050I  LOGRCDI:                        10
PTY050I  LOGMAXLOCKTIME:                 50
PTY050I  LOGMAXDELAY:                   200
PTY050I  LOGSHMPCT:                      15
PTY050I  LOGFILTER:                       6
PTY050I  PEPSTIME:                      240
PTY050I  USERNAME:       (USERID,GROUPNAME)
PTY050I  FROM_CODEPAGE:                1047
PTY050I  TO_CODEPAGE:                   923
PTY003I  Protegrity Data Security Platform Cryptographic Service
PTY003I  Manager startup complete.

PTY041I  Protegrity PEP Server startup in progress.

PTY042I  Protegrity PEP Server is started.
PTY031I  Protegrity Data Security Platform policy updated.
```

The following table describes types of informational messages displayed.

Table 4-1: Message Types

Message Type	Description
PTY050I	These messages should agree with the values assigned in the *PTYPARM* file.
PTY607I	This message indicates that hardware encryption and decryption will be used. If your site is running on an older system or if the cryptographic hardware is not enabled, then you will receive a slightly different message stating that software encryption will be done.
PTY031I	This message is the signal that the cryptographic services are available and ready for use. This message will be repeated whenever *PTYSPEPS* updates the policy. This happens during deployment of a new policy and whenever a time interval expires within the policy, i.e., at the beginning and end of the work day as defined in the policy. Note that the *PTYSPEPS* ended message during this time is normal and expected. *PTYSPEPS* uses *BPXBATCH* to start the PEP server daemon. Once the daemon is started, the MVS task is finished and ends, but the daemon continues to run. Only if the PTY031I message does not occur is there a problem and you will then need to follow debugging procedures.

You can verify the installation by starting the *PTYCSRV* procedure which will start the Protegrity Cryptographic Services Manager Task and the other servers.

> **Note:** Use the procedure customized during installation. All services can be stopped by stopping the *PTYCSRV* procedure from an operator console.

4.5.1 *PTYCSRV* Return Messages

The *PTYCSRV* may issue several messages that should be monitored by system automation.

The following table describes the *PTYCSRV* messages.

Table 4-2: PTYCSRV Messages

Message ID	Definition	Issued when...
PTY031I	Policy updated.	The security policy is loaded and available for use. Once this message is issued, ciphering operations can begin.
PTY045I	PEP Server startup timed out.	The handshake between the Open MVS PEP Server and *PTYCSRV* fails. This means that the security policy is not loaded and ciphering cannot happen.
PTY052I	The maximum number of allowed buffer queue blocks specified on the *LOGMQB* start-up parameter has been reached. Log records may be lost.	The last *LOGQBS* buffer is allocated. Normally this means that for some reason the Open MVS PEP Server is not sending log records at all or quickly enough to the ESA appliance. Check the pepserver.log in the PEP Server data sub-directory
PTY053I	Log records have been lost due to insufficient buffer queue storage. This message may precede or follow PTY052I. If it precedes PTY052I by several seconds, then increase *LOGQBS* in *PTYPARM* before the next restart of *PTYCSRV*.	A log message buffer is completely full and another log record is created by *PTYCSRV* can allocate another buffer or the maximum number of buffers has been reached.
PTY080W	The security policy has been locked by the PEP Server because logging disk space has been exhausted. Cryptographic services are suspended until the policy is unlocked. Refer to the pepserver.log for details.	The *diskfullalert* value in the Open MVS PEP Server `pepserver.cfg` is reached. No further ciphering can take place until PTY081I message is issued. There is mostly likely an issue with the PEP Server sending log records to the ESA appliance.
PTY082W	The disk space available to the PEP Server for logging is less than the specified minimum. Refer to the `pepserver.log` for details. If available disk space is exhausted, then the policy may be locked or logging may be suspended.	The threshold value the Open MVS PEP Server `pepserver.cfg` is reached. Further logging will still occur, but the reasons why the threshold has been reached should be investigated. If nothing is done, then the diskfullalert may be reached and this can cause ciphering problems as mentioned in PTY080W.
PTY083I	The disk space shortage for PEP Server logging has been relieved. Refer to the pepserver.log for details.	Disk space becomes available for continued logging.
PTY084W	The disk space available to the PEP Server for logging has been exhausted. Refer to the `pepserver.log` for details. Logging is suspended until sufficient disk space is available. In addition, refer to the message PTY080W.	The *diskfullalert* value is reached. The *diskfullaction* parameter in `pepserver.cfg` determines whether this message or PTY080W will be issued when the diskfullalert value is reached.

Message ID	Definition	Issued when...
PTY085I	The disk space shortage for PEP Server logging has been relieved. Refer to the `pepserver.log` for details. Logging is resumed.	The disk space problem for the Open MVS PEP Server is resolved. It is the inverse of PTY084W.
PTY120W	Message PTY120W is returned every 24 hours until the policy in shared memory is updated.	1. The policy that exists in the shared memory has not been updated in 24 hours. 2. When ESA is suspended, or powered off.
PTY121E	NOCST is disabled in v6.5.2 and later versions, due to which software encryption or decryption for 3DES is disabled.	Software encryption or decryption for 3DES is not supported with *PTYPFPS*.
PTY124W	Warns the system about the policy issue.	Policy refresh message is issued and there are no users in the policy.

4.6 Upgrading the z/OS Protector

This section demonstrates the process to upgrade the z/OS Protector.

To upgrade the z/OS protector:

1. Keep the existing version running if required.
2. Install the PEP server into a new volume or to a new subdirectory on the current volume. This includes loading the certificates and keys.
3. Install the new MVS libraries into new libraries and authorize the new libraries.
4. Change the communications ID on the new PEP server to some other communications ID in the `pepserver.cfg` file.
5. Customize the *PTYxxxx* JCL sample to point to the new PEP server and new communications ID. It is recommended that you use new procedure names for this step.
6. Start the new PEP server and deploy the policy. Both versions of the PEP server can be run at the same time.
7. Verify that the policy is deployed and that roles and token elements (if any) have been received. Refer to the `pepserver.log` file for this information.
8. At a convenient time, stop the old Cryptographic Services Manager and, if using the Database Protector, any DB2 regions using the database manager.
9. If using the Database Protector, replace the old version library DD statement with the new one in the *xxxxDBM1* JCL. It is recommended to just comment out the old one at this point. Restart DB2.
10. Start the new Cryptographic Services Manager.

Chapter 5

Application Protector on Mainframe z/OS

Protegrity provides the Cryptographic Service Interfaces (CSx and CXx) as Application Protector solution on Mainframe z/OS platform.

The Cryptographic Service Interface for z/OS provide multiple application programming interfaces that can be called from application programs to protect and unprotect file records or fields and any other application data. All the protect and unprotect requests are authenticated in accordance with your security policy defined in the Member Source tab of Policy Management.

The cryptographic service interfaces use a policy data element name specified in the call and the operating system USERID to validate the protection or unprotection defined in the security policy. They can use the same data element name that is employed to protect DB2 or other data or they can work with a different data element. Some of the interfaces are designed to be called by Language Environment programs and some are designed to be called by Assembler programs.

The *PTYPCSI*, *PTYPCXI* and *PTYPSLI* are tolerant of but do not conform to the IBM Language Environment. If you want to call the interfaces from a C or C++ program, you must specify **`#pragma …(OS)`**.

The *PTYPCSL*, *PTYPCXL*, and *PTYPSLL* conform to the Language Environment. You must specify **`#pragma …(OS)`** for all these programs when calling them from C or C++.

To use any of these programming interfaces, call the appropriate routine from an application program passing the data to be protected or unprotected and the related control data including the policy data element name used to authenticate the access.

> **Note:**
> Protegrity Data Security Platform does not make use of the existing IBM software product, z/OS Integrated Cryptographic Service Facility (ICSF). It is not required to use any of the supported cryptographic hardware and to perform software-only encryption and decryption.

5.1 Installing and Customizing the Application Protector on z/OS

This section explains the installation of the Application Protector solutions on Mainframe z/OS. It provides a high-level setup overview as well as post-installation verification steps.

5.1.1 Installing the Cryptographic Service Interfaces on Mainframe z/OS

The procedures in this section explain in high level how to install the CSs and CXx Interfaces.

To install the Cryptographic Service Interfaces on Mainframe z/OS:

1. Install the PEP server. For more information about installing the PEP server, refer to the section *Installing the PEP Server*.
2. Install the Cryptographic Services. For more information about installing the Cryptographic Services, refer to the section *Installing the Cryptographic Server*.
3. Install the Cryptographic Service Interface libraries.

   The `PTYxxxxx.APLOAD`, `PTYxxxxx.APSAMP`, `PTYxxxxx.APSCNTL`, and `PTYxxxxx.APSLOD` files are restored using `LODxxxA.jcl`

APLOAD	Library containing the API executables. These are modules to be linked to user application programs. The linking maybe done dynamically or statically. Refer to the appropriate Programming Language manual for how to do the dynamic linking. Are standard load module libraries. These are PDS libraries but may be PDS-E if desired.
APSAMP	Library containing the API samples. Is a RECFM F LRECL 80 JCL type library and may be either PDS or PDS-E.
APSCNTL	Optional library that contains several of the sample programs linked with the appropriate API executable. Contains the compiled and linked versions of the sample programs in APSAMP. This is a PDS library but may be PDS-E if desired.
APSLOD	Optional library containing the AP Standard and AP Client executables. The H files describing the functions are included in the Open MVS install.

4. Activate the Protegrity Data Security Platform when the Cryptographic Services Interfaces are installed. For more information about activating the Protegrity Data Security Platform, refer to the section *Activating Protegrity Data Security Platform*.
5. Deploy a test policy. For more information about deploying a test policy, refer to the section *Deploying a Test Policy*.
6. Verify the PEP server installation. For more information about verifying the installation of the PEP server, refer to the section *Verifying the Installation*.
7. Verify the protector installation. For more information about verifying the installation of the Application Protector on z/OS, refer to the section *Verifying Installation*.

5.1.2 Verifying Installation

This section outlines the process of verifying the Application Protector installation on z/OS.

To verify the installation:

1. Start the PTYCSRV which will start the PEP server automatically.
2. Define or update a policy for the installation verification test using Policy Management from the ESA.
   - For all APIs, use either CUSP or non-CUSP algorithms, or tokens. Note that the application needs to check the output length of the data (protected data for protection or clear-text data for unprotection) as the length of the data will change if non-CUSP or non-length preserving tokens are used. The application should also handle the error that may be returned showing that the output buffer size is not large enough.
   - On encryption, the Non-CUSP algorithms cause the output data to be larger than the input data, due to the padding added to the clear text data before encryption. However, we remove the padding before decryption, and the Non-CUSP algorithms causes the output data to be smaller than the output data. For more details, refer to the section *CUSP* in the *Protection Methods Reference Guide 9.2.0.0*.

3. Create the input test data by:
   RECEIVE INDSN('PTYxxxx.APSAMP(CFUDATAF)') DSN('PTYxxxx.APSAMP. CFUDATAF')
4. For *PTYPSLL/SLI*, customize the job in ***PTYxxxx.APSCNTL*** (*PTYJSL1*- compile & *PTYRSL1* - run) as directed by the inline commentary and submit the job.
5. Examine the results.

   Get the output of the PRINT step of the *PTYRSL1* job and verify ***INDATA***, ***MASTER***, and ***OUTDATA*** files. The ***INDATA*** files consists of input data, the ***MASTER*** file consists of cipher text, whereas the ***OUTDATA*** files consists of clear text.

   For further verification, look out for a confirmation message in the ***IEBCOMPR*** utility, such as, ***"END OF JOB-TOTAL NUMBER OF RECORDS COMPARED = 00001000"***. This indicates that the ***OUTDATA*** file, that includes the clear text (decrypted data), is same as the original input data in the ***INDATA*** file.

> **Note:** For more details on sample programs, refer to the section *Appendix E: Compilation of Programs*.

5.2 Stateless Interfaces (APIs): *PTYPSLI*, *PTYPSLL*, and *PTYPSLC*

The Interface modules *PTYPSLL*, *PTYPSLI* and *PTYPSLC* are recommended Stateless Interfaces.

The *PTYPSLI* is not Language Environment compliant, but is compatible with it. It uses the z/OS MACROs for storage.

The *PTYPSLL* is Language Environment compliant and uses the LE components for storage.

The *PTYPSLC* is Language Environment complaint and uses LE components for storage just like PTYPSLL.

- It checks for an additional parameter after the two external IV parameters. The parameter provides an alternate user ID to access the APIs.

> **Caution:** From v7.0 onwards, Protegrity has deprecated the support of PTYPCSL, and PTYPCSI Stateless Interfaces.

5.2.1 COBOL Parameters to call Stateless Interfaces

The *PTYPSLL*, *PTYPSLI* and *PTYPSLC* are called from COBOL test programs.

The parameters to be passed to the API from the calling program are listed as follows:

- Function to perform
- Length of the Clear Text
- Clear text / pointer to clear text
- Length of the Cipher Text
- Cipher text / pointer to cipher text
- Data-element to be used
- Return-Code from API
- Reason-Code from API
- Length of the external initialization vector
- Buffer containing the external initialization vector
- Alternate user (*PTYPSLC* only)

5.2.2 Summary of all Parameters for *PTYPSLI*, *PTYPSLL*, and *PTYPSLC*

The parameters are present in the order that they need to be specified by the caller. The *PTYCCSIP* member of *PTYxxxx.APSAMP* is a COBOL COPY section for these parameters. The names used for the parameters are the ones defined in *PTYCCSIP*.

The following table describes these parameters.

Table 5-1: Parameters in PTYCCSIP

Parameter	Format/PIC	How to use...	Remarks
CSI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Length of the clear text data.	For encryption, this is the size of the input buffer. For decryption, this is the size of the output buffer and will be set to the actual size of the data.
Clear text	clear text data	Buffer for the clear text.	For encryption, this is the input buffer. For decryption, it is the output buffer. The caller must allocate the output buffer before calling API.
CSI-Cipher-Text-Length	fullword integer / PIC 9(9) Binary	Length of the encrypted data.	For encryption, this is the size of the output buffer. For decryption, this is the size of the input buffer.
Encrypted text	cipher text data	Buffer for the encrypted data.	For encryption, this is the output buffer. For decryption, this is the input buffer. The caller must allocate the output buffer before calling API.
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Success or Failure from the API.	Zero return equals success. Non-zero indicates a failure.
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CSI-Return Code.	The most common ones are documented in section *Return and Reason Codes*.
CSI-Function	one character / PIC X	This parameter defines whether encryption or decryption should be done.	Based on the value that is passed to the parameter, respective operation is triggered.
CSI-DPS-Data-Element	56 characters / PIC X(56)	Policy data element name to be used to authenticate access to the data.	This data element name needs to be in the same case as the data element name was defined in the policy, i.e., if the policy has the data element name defined as mixed (PTY_ivp_Test_Data) then the value of this must be “PTY_ivp_Test_Data”. This can be a data item or a literal but the literal should be 56 characters with trailing blanks. If using Query DDE, then this must be a data item, as the default data element will be returned in this item.
CSI-External-IV-Length	fullword integer / PIC 9(9) Binary	Length of the external initialization vector.	Length of the external initialization vector. This parameter is optional.
External-IV Text	External initialization vector	Buffer containing the external initialization vector.	This is the value of external IV.

Parameter	Format/PIC	How to use...	Remarks
			This parameter is optional. If specified, then a length must also be specified.
CSI-ALT-ID	8 characters / PIC X(8)	Optional user provides an alternate user Id to be used in lieu of the logon user	This parameter is only used with *PTYPSLC* Interface. **Note:** If this is used in a PCI standards' context, use of alternate user ID may be in scope for any PCI auditor review.

Note:
For data protection, if the policy data element specifies a non-length preserving algorithm, that includes encryption with specified key-id, then the protected data length is returned in the CSI-Cipher-Text-Length. Likewise, the unprotected data length is returned in the CSI-Clear-Text-Length on an unprotecting operation. For protected data, the returned length should be maintained with the data so that the data is properly unprotected. The application should also handle the data that is too short. Error can be returned with protection, when the output buffer (CSI-Cipher-Text-Length) is too short to contain the protected value.

Note:
The external IV is ignored during encryption.

5.2.2.1 Functions performed by Stateless Interfaces

The parameters are present in the order that they need to be specified by the caller. This must be the first call to *PTYPSLI*, *PTYPSLL*, or *PTYPSLC*. Ensure that the clear text values and the cipher text values are always specified in the same order in the parameter. The function code determines whether encryption or decryption is done.

Any function that encrypts, decrypts, or queries follows the given syntax.

CALL the "Stateless Interfaces" using

- CSI-Function
- CSI-Clear-Text-Length
- WS-Clear-Text
- CSI-Cipher-Text-Length
- WS-Cipher-Text
- CSI-DPS-Data-Element
- CSI-Return-Code
- CSI-Reason-Code
- CSI-External-IV Length
- WS-EIV-Text
- ALT-ID

5.2.2.1.1 Encrypt Functions

The encrypt functions include functions supported for encryption and software encryption.

The following functions are explained in this section:

- *CSI-Encrypt Function*
- *CSI- Encrypt SFW*

5.2.2.1.1.1 CSI-Encrypt Function

This function is used for encryption.

Parameters

Parameter	Type/PIC	Description
CSI-Function	one character / PIC X	Value is 1.
CSI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Length of the clear text data or the size of the input buffer.
Clear text	clear text data	Buffer for the clear text or the input buffer.
CSI-Cipher-Text-Length	fullword integer / PIC 9(9) Binary	Length of the encrypted data or the size of the output buffer.
Encrypted text	cipher text data	Buffer for the encrypted data or the output buffer.
CSI-DPS-Data-Element	56 characters / PIC X(56)	Policy data element name to be used for data encryption.
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CSI-Return Code.

Result

The Return code and Reason Code will be returned after the call, signifying the successful or unsuccessful protection.

The variable defined for the encrypted text will have the corresponding protected data.

Parameter	Type/PIC	Description
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CSI-Return Code.
CSI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Calculated length of the encrypted text.
Clear text	clear text data	Encrypted text after the encryption/ tokenization.

5.2.2.1.1.2 CSI-Encrypt SFW

This function is used for software encryption. The Encrypt-SFW and Encrypt provide the same result, except for the improved performance for CPU and clock time in case of Encrypt.

Parameters

Parameter	Type/PIC	Description
CSI-Function	one character / PIC X	Value is 2.

Parameter	Type/PIC	Description
CSI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Length of the clear text data or the size of the input buffer.
Clear text	clear text data	Buffer for the clear text or the input buffer.
CSI-Cipher-Text-Length	fullword integer / PIC 9(9) Binary	Length of the encrypted data or the size of the output buffer.
Encrypted text	cipher text data	Buffer for the encrypted data or the output buffer.
CSI-DPS-Data-Element	56 characters / PIC X(56)	Policy data element name to be used for data encryption.
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure. Additional information about the CSI-Return Code.

Result

Parameter	Type/PIC	Description
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CSI-Return Code.
CSI-Cipher-Text-Length	fullword integer / PIC 9(9) Binary	Calculated length of the encrypted text.
Encrypted text	cipher text data	Encrypted text after encryption/ tokenization.

5.2.2.1.2 Decrypt Functions

The decrypt functions include functions supported for decryption and software decryption.

The following functions are explained in this section:

- *CSI- Decrypt Function*
- *CSI- Decrypt SFW*

5.2.2.1.2.1 CSI-Decrypt Function

This function is used for decryption.

Parameters

Parameter	Type/PIC	Description
CSI-Function	one character / PIC X	Value is 3.
CSI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Length of the decrypted text data or the size of the output buffer.
Clear text	clear text data	Buffer for the decrypted text or the output buffer.
CSI-Cipher-Text-Length	fullword integer / PIC 9(9) Binary	Length of the encrypted data or the size of the input buffer.
Encrypted text	cipher text data	Buffer for the encrypted data or the input buffer.

Parameter	Type/PIC	Description
CSI-DPS-Data-Element	56 characters / PIC X(56)	Policy data element name to be used for data decryption.
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CSI-Return Code.

Result

The Return code and Reason Code will be returned after the call, signifying the successful or unsuccessful unprotection.

The variable defined for the decrypted text will have the corresponding decrypted data.

Parameter	Type/PIC	Description
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CSI-Return Code.
CSI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Calculated length of the decrypted text.
Clear text	clear text data	Decrypted text after the decryption/ detokenization.

5.2.2.1.2.2 CSI-Decrypt SFW

This function is used for software decryption. The Decrypt-SFW and Decrypt provide the same result, except for the improved performance for CPU and clock time in case of Decrypt.

Parameters

Parameter	Type/PIC	Description
CSI-Function	one character / PIC X	Value is 4.
CSI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Length of the decrypted text data or the size of the output buffer.
Clear text	clear text data	Buffer for the decrypted text or the output buffer.
CSI-Cipher-Text-Length	fullword integer / PIC 9(9) Binary	Length of the encrypted data or the size of the input buffer.
Encrypted text	cipher text data	Buffer for the encrypted data or the input buffer.
CSI-DPS-Data-Element	56 characters / PIC X(56)	Policy data element name to be used for data decryption.
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CSI-Return Code.

Result

Parameter	Type/PIC	Description
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success.

Parameter	Type/PIC	Description
		Non-zero indicates a failure.
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CSI-Return Code.
CSI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Calculated length of the decrypted text.
Clear text	clear text data	Decrypted text after the decryption/ detokenization.

5.2.2.1.3 CSI-Query DDE

This function performs the sanity check for the protector.

The parameters are the same as for encryption or decryption calls except for the CSI-DPS-Data-Element parameter. In the Query DDE case, the input CSI-DPS-Data-Element is the policy name. The default data element for the specified policy name will be returned in the CSI-DPS-Data-Element field. In this case, the CSI-DPS-Data Element field must NOT be a literal and must be a data item. If the specified policy name does not exist, you will receive a return code of 12 and a reason of 120128558 (x'0729042E').

Parameters

Parameter	Type/PIC	Description
CSI-Function	one character / PIC X	Value is 9.
CSI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Length of the clear text data. For encryption, this is the size of the input buffer. For decryption, this is the size of the output buffer and will be set to the actual size of the data.
Clear text	clear text data	Buffer for the clear text. For encryption, this is the input buffer, and for decryption it is the output buffer. The caller must allocate the output buffer before calling API.
CSI-Cipher-Text-Length	fullword integer / PIC 9(9) Binary	Length of the encrypted data. For encryption, this is the size of the output buffer. Depending upon the encryption algorithm, the length could be longer than the clear text length. If the specified length in not long enough, an error will be returned. For decryption, this is the size of the input buffer.
Encrypted text	cipher text data	Buffer for the encrypted data. For encryption, this is the output buffer, and for decryption this is the input buffer. The caller must allocate the output buffer before calling API.
CSI-DPS-Data-Element	56 characters / PIC X(56)	Policy name.
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CSI-Return Code.

Result

Parameter	Type/PIC	Description
CSI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.

Parameter	Type/PIC	Description
CSI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CSI-Return Code.
CSI-DPS-Data-Element	56 characters / PIC X(56)	Default Data element of the policy.

Note:
Samples to explain how *PTYPSLL*, *PTYPSLI*, or *PTYPSLC* can be called from COBOL to perform various functions, are mentioned in section *Appendix E- z/OS Application Protector Samples*.

5.2.3 Return and Reason Codes

This section lists the Cryptographic Service Interface (CSI) return codes and reason codes.

The Cryptographic Service Interface (CSI) return codes and reason codes have a 4 bytes binary format. Some errors may include a PTY105E message with more information.

Note: The CSI may ABEND if the return code and reason code parameters cannot be accessed.

The following table describes the return and reason codes:

Error	Return Code	Dec.	Reason Code	Dec.
Access denied	X'00000004'	4	X'00000006'	6
Access denied - Unknown user	X'00000004'	4	X'00000001'	1
Access denied - Unknown data element	X'00000004'	4	X'00000002'	2
Access denied - User has no privileges for data element	X'00000004'	4	X'00000003'	3
Access denied - User has no privileges for data element for the current time period	X'00000004'	4	X'00000004'	4
ALT-ID value is mentioned without externalIV parameters	X'0000 0008'	8	x'50000000'	1342177280
Login user is in the role and ALT-ID is empty	x'08000000'	8	x'51000000'	1358954496
Function address null	X'0000000C'	12	X'0000000A'	10
Clear text length address null	X'0000000C'	12	X'0000000B'	11
Clear text address null	X'0000000C'	12	X'0000000C'	12
Cipher text length address null	X'0000000C'	12	X'0000000D'	13
Cipher text address null	X'0000000C'	12	X'0000000E'	14
Data element address null	X'0000000C'	12	X'0000000F'	15
Data Protector not active	X'0000000C'	12	X'00000011'	17
Data Protector not active	X'0000000C'	12	X'00000017'	23
Product not active	X’0000000C’	12	X’0000001E’	30
Invalid function code	X’0000000C’	12	X’00000010’	16

Error	Return Code	Dec.	Reason Code	Dec.
Policy not ready	X’0000000C’	12	X’0000001F’	31
Plain text length zero	X’0000000C’	12	X’00000029’	41
Clear text length zero	X’0000000C’	12	X’00000029’	41
Cipher text length zero	X’0000000C’	12	X’0000002A’	42
Cipher text length less than plain text length	X’0000000C’	12	X’0000002B’	43
Cipher text length less than clear text length	X’0000000C’	12	X’0000002B’	43
Query null data element	X’0000000C’	12	X’0000002D’	45
Invalid checksum	X’0000000C’	12	X’0000002F’	47
Invalid padding	X’0000000C’	12	X’00000030’	48
Invalid IV	X’0000000C’	12	X’00000031’	49
Cipher text area too small	X’0000000C’	12	X’00000032’	50
Plain text area too small	X’0000000C’	12	X’00000033’	51
Checksum validation failure	X’0000000C’	12	X’00000034’	52
Unsupported algorithm	X’0000000C’	12	X’00000058’	88
Unknown algorithm	X’0000000C’	12	X’0000005D’	93
Default data element not supported	X’0000000C’	12	X’0000005E’	94
Abend percolated	X’0000000C’	12	X’000000FB’	251
Peps policy locked	X’0000000C’	12	X'072C0000'	120324096

Note: Some errors will also include a PTY105E message that has more information about the issue.

CSI may ABEND if the return code and reason code parameters cannot be accessed.

Error	User Abend Code
Parmlist address null	001
Return code address null	002
Reason code address null	003
Return code parameter omitted	005
Reason code parameter omitted	006

5.3 Stateful Interfaces: *PTYPCXI* and *PTYPCXL*

The Interface modules *PTYPCXI* and *PTYPCXL* are recommended Stateful Interfaces.

The Interface modules *PTYPCXI* and *PTYPCXL* are stateful. This means that they maintain information from previous calls to allow simplifying future processing. This also means that unlike the stateless interfaces, there are multiple function calls to use *PTYPCXI* or *PTYPCXL*.

5.3.1 COBOL Prameters to call Stateful Interfaces

The *PTYPCXI* and *PTYPCXL* are called from COBOL test programs.

The parameters to be passed to the API from the calling program are listed as follows:

- Function to perform

- Length of the Clear Text
- Clear text / pointer to clear text
- Length of the Cipher Text
- Cipher text / pointer to cipher text
- Data-element to be used
- Return-Code from API
- Reason-Code from API
- Length of the external initialization vector
- Buffer containing the external initialization vector

5.3.2 Summary of all Parameters for *PTYPCXI* and *PTYCXL*

This section discusses about the summary of all parameters for *PTYPCXI* and *PTYCXL*.

There are three calls to *PTYPCXI* and *PTYCXL*. These calls are specified via the CXI-Function. The parameters are different for each call. The parameters are present in the order that they need to be specified by the caller. The *PTYCCXIP* member of *PTYxxxx.APSAMP* is a COBOL COPY section for these parameters. The names used for the parameters are the ones defined in *PTYCCXIP*.

Connect and disconnect are pairs. A connect call is followed by one or more encrypt or decrypt calls with a disconnect call at the end of the logic.

Table 5-2: Parameters in PTYCCXIP

Parameter	Type	Description
CXI-Function	one character	Value is 1 for encrypt and value is 3 for decrypt.
CXI-Handle	8 characters (PIC X(8) for COBOL)	Value is returned by the connect call and must be passed ASIS on each subsequent encrypt, decrypt or disconnect call. The calling program should NOT change this value.
CXI-Clear-Text-Length	fullword integer	Length of the clear text data. For decryption, this is the size of the output buffer and will be set to the actual size of the data. For encryption, this is the size of the input buffer.
Clear text	clear text data	Buffer for the clear text. For encryption, this is the input buffer, and for decryption it is the output buffer. The caller must allocate the output buffer before calling PTYPCXI. Also, refer to the section *Current Data Element Key ID Query Configuration*.
CXI-Cipher-Text-Length	fullword integer	Length of the encrypted data. For encryption, this is the size of the output buffer. Depending upon the encryption algorithm, the length may need to be longer than the clear text length. If the specified length in not long enough, then an error will be returned. For decryption, this is the size of the input buffer.
Encrypted text	cipher text data	Buffer for the encrypted data. For encryption, this is the output buffer, and for decryption this is the input buffer. The caller must allocate the output buffer before calling PTYPCSI. Also, refer to the section *Current Data Element Key ID Query Configuration*.

Parameter	Type	Description
CXI-Return-Code	fullword integer	Zero return equals success. Non-zero indicates a failure and the reason code will have more information.
CXI-Reason-Code	fullword integer	The most common ones are documented in the section *Return and Reason Codes*.
CXI-External-IV-Length	fullword integer	Length of the external initialization vector. This parameter is optional.
External-IV Text	External initialization vector	This is the value of external IV. This parameter is optional. If specified, then a length must also be specified.

Note:

For data protection, if the policy data element specifies a non-length preserving algorithm, that includes encryption with specified key-id, then the protected data length is returned in the CXI-Cipher-Text-Length. Likewise, the unprotected data length is returned in the CXI-Clear-Text-Length on an unprotecting operation. For protected data, the returned length should be maintained with the data so that the data is properly unprotected. The application should also handle the data that is too short. Error can be returned with protection, when the output buffer (CXI-Cipher-Text-Length) is too short to contain the protected value.

Note:

The external IV is ignored during encryption.

5.3.2.1 Functions performed by Stateful Interfaces

This section demonstrates the functions performed by *PTYPCXI* or *PTYPCXL*.

The following are the functions:

- *Connect Configuration*
- *Encrypt Configuration*
- *Decrypt Configuration*
- *Disconnect Configuration*
- *Default Data Element (DDE) Configuration*
- *Current Data Element Key ID Query Configuration*

5.3.2.1.1 Connect Configuration

This function is used for connect configuration.

The parameters are present in the order that they need to be specified by the caller. This must be the first call to *PTYPCXI* or *PTYPCXL*. The following table describes the connect parameters,

Parameters

Parameter	Type/PIC	Description
CXI-Function	one character	Value is 7.
CXI-Handle	8 characters (PIC X(8) for COBOL)	Value is returned by the connect call and must be passed on each subsequent encrypt, decrypt or disconnect call. The calling program should NOT change this value.

Parameter	Type/PIC	Description
CXI-DPS-Data-Element	56 characters	Policy data element name to use to authenticate access to the data. This needs to be in the same case as the data element name was defined in the policy, i.e., if the policy has the data element name defined as mixed (PTY_ivp_Test_Data) then the value of this must be “PTY_ivp_Test_Data”. This can be a data item or a literal but the literal should be 56 characters with trailing blanks.

Result

The Return code and Reason Code will be returned after the call, signifying the successful or unsuccessful protection.

The variable defined for the encrypted text will have the corresponding protected data.

Parameter	Type/PIC	Description
CXI-Return-Code	fullword integer	Zero return equals success. Non-zero indicates a failure and the reason code will have more information.
CXI-Reason-Code	fullword integer	The most common ones are documented in the section *Return and Reason Codes*.

5.3.2.1.2 Encrypt Configuration

This function is used for encryption.

The parameters are present in the order that they need to be specified by the caller. This must be the first call to *PTYPCXI* or *PTYPCXL*.

Parameters

Parameter	Type/PIC	Description
CXI-Function	one character	Value is 1.
CXI-Handle	8 characters (PIC X(8) for COBOL)	Value is returned by the connect call and must be passed ASIS on each subsequent encrypt, decrypt or disconnect call. The calling program should NOT change this value.
CXI-Clear-Text-Length	fullword integer	Length of the clear text data. For encryption, this is the size of the input buffer.
Clear text	clear text data	Buffer for the clear text. For encryption, this is the input buffer. The caller must allocate the output buffer before calling PTYPCXI. Also, refer to section *Current Data Element Key ID Query Configuration*.
CXI-External-IV-Length	fullword integer	Length of the external initialization vector. This parameter is optional.
External-IV Text	External initialization vector	This is the value of external IV. This parameter is optional. If specified, then a length must also be specified.

Result

The Return code and Reason Code will be returned after the call, signifying the successful or unsuccessful protection.

The variable defined for the encrypted text will have the corresponding protected data.

Parameter	Type/PIC	Description
CXI-Return-Code	fullword integer	Zero return equals success. Non-zero indicates a failure and the reason code will have more information.
CXI-Reason-Code	fullword integer	The most common ones are documented in the section *Return and Reason Codes*.
CXI-Cipher-Text-Length	fullword integer / PIC 9(9) Binary	Calculated length of the encrypted text.
Encrypted text	cipher text data	Encrypted text after encryption/ tokenization.

5.3.2.1.3 Decrypt Configuration

This function is used for decryption.

The parameters are present in the order that they need to be specified by the caller. This must be the first call to *PTYPCXI* or *PTYPCXL*.

Parameters

Parameter	Type/PIC	Description
CXI-Function	one character	Value is 3.
CXI-Handle	8 characters (PIC X(8) for COBOL)	Value is returned by the connect call and must be passed ASIS on each subsequent encrypt, decrypt or disconnect call. The calling program should NOT change this value.
CXI-Cipher-Text-Length	fullword integer	Length of the encrypted data. For decryption, this is the size of the input buffer. Depending upon the encryption algorithm, the length may need to be longer than the clear text length. If the specified length in not long enough, then an error will be returned.
Encrypted text	cipher text data	Buffer for the encrypted data. For decryption, this is the input buffer. The caller must allocate the output buffer before calling PTYPCSI. Also, refer to section *Current Data Element Key ID Query Configuration*.
CXI-External-IV-Length	fullword integer	Length of the external initialization vector. This parameter is optional.
External-IV Text	External initialization vector	This is the value of external IV. This parameter is optional. If specified, then a length must also be specified.

Result

The Return code and Reason Code will be returned after the call, signifying the successful or unsuccessful protection.

The variable defined for the decrypted text will have the corresponding protected data.

Parameter	Type/PIC	Description
CXI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.
CXI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CXI-Return Code.
CXI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Calculated length of the decrypted text.
Clear text	clear text data	Decrypted text after the decryption/ detokenization.

5.3.2.1.4 Disconnect Configuration

This function is used for disconnect configuration.

The parameters are present in the order that they need to be specified by the caller. This must be the first call to *PTYPCXI* or *PTYPCXL*.

The following table describes the disconnect codes.

Parameters

Parameter	Type/PIC	Description
CXI-Function	one character	Value is 8 for disconnect.
CXI-Handle	8 characters (PIC X(8) for COBOL)	Value is returned by the connect call and must be passed on each subsequent encrypt, decrypt or disconnect call. The calling program should NOT change this value. Disconnect will zero the value.

Result

The Return code and Reason Code will be returned after the call, signifying the successful or unsuccessful protection.

The variable defined for the encrypted text will have the corresponding protected data.

Parameter	Type/PIC	Description
CXI-Return-Code	fullword integer	Zero return equals success. Non-zero indicates a failure and the reason code will have more information.
CXI-Reason-Code	fullword integer	The most common ones are documented in the section *Return and Reason Codes*.

Caution: If the *DISCONNECT* function is not called while using *PTYPCXI* or *PTYPCXL*, then successful audit logs will not be generated, and unsuccessful logs will still be generated.

5.3.2.1.5 Default Data Element (DDE) Configuration

This function performs the sanity check for the protector.

The parameters are the same as for encryption or decryption calls except for the CXI-DPS-Data- Element parameter. In the Query DDE case, the input CXI-DPS-Data-Element is the policy name. The default data element for the specified policy name will be returned in the CXI-DPS-Data-Element field. In this case, the CXI-DPS-Data Element field must NOT be a literal and must be a data item. If the specified policy name does not exist, you will receive a return code of 12 and a reason of 120128558 (x'0729042E').

Parameters

Parameter	Type/PIC	Description
CXI-Function	one character / PIC X	Value is 9.
CXI-Clear-Text-Length	fullword integer / PIC 9(9) Binary	Length of the clear text data. For encryption, this is the size of the input buffer, whereas, for decryption, this is the size of the output buffer and will be set to the actual size of the data.
Clear text	clear text data	Buffer for the clear text. For encryption, this is the input buffer, whereas, for decryption it is the output buffer. The caller must allocate the output buffer before calling API.
CXI-Cipher-Text-Length	fullword integer / PIC 9(9) Binary	Length of the encrypted data. For encryption, this is the size of the output buffer. Depending upon the encryption algorithm, the length could be longer than the clear text length. If the specified length in not long enough, an error will be returned. For decryption, this is the size of the input buffer.
Encrypted text	cipher text data	Buffer for the encrypted data. For encryption, this is the output buffer, whereas, for decryption this is the input buffer. The caller must allocate the output buffer before calling API.
CXI-DPS-Data-Element	56 characters / PIC X(56)	Policy name
CXI-Return-Code	fullword integer / PIC 9(9) Binary	Zero return equals success. Non-zero indicates a failure.
CXI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CXI-Return Code.

Result

The Return code and Reason Code will be returned after the call, signifying the successful or unsuccessful operation.

The variable defined for the encrypted text will have the corresponding protected data.

Parameter	Type/PIC	Description
CXI-Return-Code	fullword integer / PIC 9(9) Binary	Zero equals success. Non-zero indicates a failure

Parameter	Type/PIC	Description
CXI-Reason-Code	fullword integer / PIC 9(9) Binary	Additional information about the CXI-Return Code.
CXI-DPS-Data-Element	56 characters / PIC X(56)	Default Data element of the policy.

Note:

Samples to explain how *PTYPCXI* and *PTYPCXL* can be called from COBOL to perform various functions, are mentioned in section *Appendix E- z/OS Application Protector Samples*.

5.3.2.1.6 Current Data Element Key ID Query Configuration

This section discusses about Current Data Element Key ID Query configuration.

PTYPCXI and *PTYPCXL* require no specific field for Key ID purposes. If the encrypted data length is not large enough to hold the encrypted field including any padding, Key ID, checksum, or initialization vector, then a **Cipher text length too short** error will be returned.

Note:

Samples to explain how *PTYPCXL/PTYPCXI* can be called from COBOL to perform various functions, are mentioned in the section *Appendix E- z/OS Application Protector Samples*.

5.3.3 Return and Reason Codes

This section lists the CXI return codes and reason codes.

The CXI return codes and reason codes are 4 bytes binary format. Many of these can also be returned for the stateless interfaces. Some errors may include a PTY105E message with more information.

Error	Return Code	Dec.	Reason Code	Dec.
Incorrect data element	X’00000004’	4	X’00000002’	2
Access denied	X'00000004'	4	X'00000006'	6
Invalid input type for token	X'00000004'	4	X’000A0000’	655360
Cipher text length zero	X'0000 0008'	8	X’24000000’	603979776
Cipher text length too small	X'0000 000C'	12	X’07000032’	117440562
Function address null	X'0000 000C'	12	X'0000000A'	10
Clear text length address null	X'0000 000C'	12	X'0000000B'	11
Clear text address null	X'0000000C'	12	X'0000000C'	12
Cipher text length address null	X'0000000C'	12	X'0000000D'	13
Cipher text address null	X'0000000C'	12	X'0000000E'	14
Data element address null	X'0000000C'	12	X'0000000F'	15
Data Protector not active	X'0000000C'	12	X'00000011'	17
Data Protector not active	X'0000000C'	12	X'17000000'	385875968
Product not active	X’0000000C’	12	X’0000001E’	30
Clear text length zero	X’0000000C’	12	X’23000000’	587202560
Peps policy locked	X’0000000C’	12	X'072C0000'	120324096

Error	Return Code	Dec.	Reason Code	Dec.
Policy not found	X’0000000C’	12	X'0729042E'	120128558
Out of address spaces	X’0000000C’	12	X'150C0000'	353107968
Cipher text length too short	X’0000000C’	12	X’07320000’	120717312
Algorithm not supported	X’0000000C’	12	X’07000058’	117440600
Cipher text length too small	X’0000000C’	12	X’07000032’	117440562

CSI may ABEND if the return code and reason code parameters cannot be accessed.

Error	User Abend Code
Parmlist address null	001
Incorrect function code	002
Reason code address null	006
Return code address null	005
Codebooks size is larger than the allocated shared memory size	2449(X'0991')

Chapter 6

File Protector Solutions on Mainframe z/OS

Protegrity provides two File Protector solutions for Mainframe z/OS platform: Cryptographic File Utility (CFU) and Cryptographic Subsystem (CSS). The installation and operation of these products are explained in the following sections.

Both Cryptographic File Utility (CFU) and Cryptographic Subsystem (CSS) require the Protegrity Cryptographic Servers, the installation of which is described in the section *z/OS Product Installation*. Both the products work only with sequential non-VSAM files.

The Cryptographic File Utility (CFU) is a program similar in concept to IEBGENER (IBM mainframe utility) that inputs a file and copies the complete file to another file making the changes determined from the control records. These changes could be partial, complete encryption, or tokenization. CFU needs an unprotected file as a part of the protection process. This is an advantage in multiple job steps when there is a need to process the unprotected file prior to the final processing by reducing protection and unprotection costs.

The Cryptographic Subsystem (CSS) takes more installation effort since it requires an MVS subsystem to be defined. CSS does not require any intermediate unprotected files. It unprotects the protected file during input I/O processing and protects the output data during the I/O output processing.

> **Note:** Protegrity does not use the IBM software product, z/OS Integrated Cryptographic Service Facility or ICSF (including their encryption and decryption methods). It may use some of the ICSF hardware, but does so with Protegrity's own methods.

6.1 Installing and Customizing File Protector on z/OS

The following section explains the installation of File Protector solutions on Mainframe z/OS. It provides a high-level setup overview as well as post-installation verification steps.

The following sections describe about setting up File Protector on z/OS:

- *Setting up File Protector on Mainframe z/OS*
- *Installing and Customizing SIAM Subsystem*

6.1.1 Setting up File Protector on Mainframe z/OS

These steps explain in high level how to setup File Protector and install CFU and CSS on Mainframe z/OS.

To install File Protector on Mainframe z/OS:

1. Install the PEP server. For more information about installing the PEP server, refer to the section *Installing the PEP Server*.
2. Install the Protegrity Cryptographic Services. For more information about installing the Cryptographic Services, refer to the section *Installing the Cryptographic Server*.
3. Install CSS, CFU, or both Interfaces as explained below.
   a. The `PTYxxxx.FPAUTH`, `PTYxxxx.FPLOAD`, and `PTYxxxx.FPSAMP` files are restored using `LODxxxF.jcl`.

Library Name	Description
`PTYxxxx.FPAUTH`	This is a standard load library that contains the executable for the MVS subsystem, which is required for the Cryptographic Sub-System. This executable may be left in this library and this library can be added to the LINKLIST and the authorized library as documented below. You can also copy the executable into the `PTYxxxx.SPTYAUTH` library that is installed as part of the Protegrity cryptographic servers' installation.
`PTYxxxx.FPLOAD`	This is a standard load library that contains the executables for CFU and CSS.
`PTYxxxx.FPSAMP`	This is a RECFM F LRECL 80 JCL type library, containing the sample data and JCLs to test CFU and CSS.

4. Activate Protegrity Data Security Platform. For more information about activating the Protegrity Data Security Platform, refer to the section *Activating Protegrity Data Security Platform*.
5. Deploy a test policy. For more information about deploying a test policy, refer to the section *Deploying a Test Policy*.
6. Verify the PEP server installation. For more information about verifying the installation of the PEP server, refer to the section *Verifying the Installation*.

6.1.2 Installing and Customizing SIAM Subsystem

This section outlines how to install and customize the Protegrity CSS for z/OS.

The following sections describe about installing and customizing SIAM Subsystem:

- *Installing Dynamic CSS*
- *Installing Permanent CSS*
- *Controlling Access to CSS*

6.1.2.1 Installing Dynamic CSS

To dynamically install a new MVS subsystem, load the initialization module from an existing link listed library. It is suggested that you only use the dynamic method of CSS installation for testing. Any changes done with this method are lost at the next Initial Program Load (IPL) of z/OS.

To dynamically install the CSS SSI:

1. Copy all modules in *FPAUTH* to an existing link-listed authorized library.
   For example, `USER.LINKLIB` and `PTYxxxx.SPTYAUTH`, among others.
2. Refresh the library lookaside (LLA), issue F LLA, REFRESH on system console.
3. Dynamically add the CSS initialization module to the LPA with the following MVS command:

```
SETPROG LPA,ADD,DSNAME=USER.LINKLIB,MODULE=(SIAMINIT,SIAMOPCL,PTYPAMLT,PTYPAMEX)
```

4. Dynamically add the new subsystem with the following command:
   ```
   SETSSI ADD,SUB=SIAM,I=SIAMINIT
   ```
   The message SIA000I OK appears on the system console and the CSS is ready to be used.

6.1.2.2 Installing Permanent CSS

You must IPL the MVS system. First, you need to make the following changes to SYS1.PARMLIB:

To install the CSS SSI permanently:

1. Add `PTYxxxx.FPAUTH` to your current *PROGxx* member.
2. Add `PTYxxxx.FPAUTH` to your current linklist either in your *PROGxx* or *LNKLSTxx* member.
3. Add to existing or new *IEALPAxx* member.
   ```
   INCLUDE LIBRARY(PTYxxxx.FPAUTH)MODULES(SIAMINIT,SIAMOPCL,PTYPAMLT,PTYPAMEX)
   ```
4. Add the subsystem to your current *IEFSSNxx* member.
   ```
   SUBSYS SUBNAME(SIAM) INITRTN(SIAMINIT) /* SIAM SUBSYSTEM */
   ```
5. Perform an IPL of the system to finally install the SSI.

> **Note:** If you previously installed the SSI dynamically, then you added the *FPAUTH* modules to an existing link-listed data set. After the SSI is installed permanently, you should remove these modules from this data set.

6.1.2.3 Controlling Access to CSS

Controlling access to CSS is required only if you plan to use CSS along with MVS authorized programs, such as IDCAMS and DB2 utility.

As part of dataset OPEN process, the SIAM subsystem issues RACHECK macro for the following profiles in FACILITY class `SIAM.jsprognm.amprognm.amddname`. The components are as explained below:

- *jsprognm* – job step program name
- *amprognm* – trusted program name allowed to execute under SIAM subsystem
- *amddname* – DD name where SUBSYS parameter specified

User must have READ authority for the preceding profile to allow them to execute amprognm under SIAM subsystem. If profile does not exist, then further processing is allowed only under non-APF authorized job step. The following code is an example of JCL for defining profiles for IDCAMS and DB2 utility.

```
//* INSERT JOBCARD
//*
//* THIS BATCH JOB DEFINES THE RACF PROFILES TO PROTECT
//* SIAM SUBSYSTEM USAGE
//*
//* VERIFY OR CHANGE THE FOLLOWING BEFORE RUNNING THE JOB:
//*
//* PROFILE OWNER  : SYS1
//* AUTH USER/GROUP : PTY
//*
//*
//IKJEFT01 EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSOUT  DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 RDEFINE FACILITY SIAM.DSNUTILB.PTYPAMLT.SYSREC  +
```

```
  UACC(NONE) OWNER(SYS1) DATA('DB2 UTILITY')
 RDEFINE FACILITY SIAM.IDCAMS.PTYPAMLT.*      +
  UACC(NONE) OWNER(SYS1) DATA('AMS UTILITY')
 PERMIT SIAM.DSNUTILB.PTYPAMLT.SYSREC       +
  CLASS(FACILITY) ID(PTY) ACCESS(READ)
 PERMIT SIAM.IDCAMS.PTYPAMLT.*           +
  CLASS(FACILITY) ID(PTY) ACCESS(READ)
 SETROPTS RACLIST(FACILITY) REFRESH
/*
//
```

Note: The following steps describe how to allow access to SIAM subsystem using the Access Control Facility ACF2 (CA-ACF2) command:

1. Navigate to the **ACF2 ISPF** dialog.
2. Select the option **1.2**.
3. Run the following command:

   ```
   $KEY(SIAM) TYPE(FAC) ROLESET

   - ROLE(-) ALLOW
   ```

4. Press the PF3 key to compile and save the rule.
5. Run the following command to rebuild the FAC in memory Infostorage rules.

   ```
   F ACF2, REBUILD(FAC)
   ```

6.2 Cryptographic File Utility for z/OS

Cryptographic File Utility for z/OS is a utility for protecting and unprotecting data stored in sequential non-VSAM files.

Cryptographic File Utility for z/OS uses standard z/OS data management methods to read and write the data. The utility can be used with files written to disk or tape, and can read or write UNIX System Services files. No programs need to be written or modified to use this functionality. JCL, CLISTs, or REXX EXECs can be used to invoke the utility.

The input file can be partially protected or unprotected by specifying a starting record number and a record count for how many records are to be protected or unprotected. If DCB attributes (logical record length, block size, record format, and so on) are not specified for the output file, then the input DCB attributes are copied to the output file.

6.2.1 Executing Protegrity Cryptographic File Utility

The Cryptographic File Utility protects or unprotects non-VSAM sequential files. It uses standard z/OS file services and Protegrity cryptographic services to do protect or unprotect. Standard data definition statements point to the input file and where to place the output file. A parameter file describes whether to protect or unprotect, which record to start with, and how many records to work with.

The following sections describe about executing Protegrity Cryptographic File Utility:

- *Parameter File*
- *Input File Specifications*
- *Output File Specifications*

6.2.1.1 Parameter File

The parameter file is a standard sequential file that is pointed to by *DDNAME PTYPARM*. It may be provided inline with the invocation JCL or it may be provided as a member of a partitioned dataset.

The following table describes the valid parameters. Any record within the parameter file that starts with an asterisk (*) is a comment and is ignored. Blanks are not allowed to be embedded within the parameters.

Table 6-1: Parameters for the Parameter File

Parameter	Description
ENCRYPT=policy data element name	Either ENCRYPT or DECRYPT must be specified. Even though ENCRPYPT and DECRYPT are used as keywords, tokens may also be used.
DECRYPT=policy data element name	Either ENCRYPT or DECRYPT must be specified.
RSTART=n	Optional parameter, that specifies the record from where the encryption or decryption should begin. The first record is record 1. If no RSTART is specified, then all records in the file are encrypted or decrypted. The value of n must be a positive integer. The number can be larger than the number of records in the file, and if so, then the input file is just copied to the output file. If RSTART is specified and RCOUNT is not specified, then the remainder of the input file is ciphered as specified by ENCRYPT or DECRYPT.
RCOUNT=n	Optional parameter that specifies the number of records to cipher. The default is to cipher the entire file. The value of n must be a positive integer and is inclusive. So RSTART=25 and RCOUNT=10 then records 25 through 34 are ciphered.
PAD=SPACES/NULLS	This parameter is only valid for CUSP algorithms. It is ignored for tokens and non-CUSP encryption algorithms. The output record is padded with the designated character. For encrypting, the padding is added before encryption. For decrypting, the padding is added after the input data is decrypted. If the input and output LRECLs are equal, then no padding is done. If not specified, then NULLs (binary zero) are added when the output record is longer than the input record.
CSTART=n*	Optional parameter that specifies the column to start the encryption or decryption. The first column is column 1. If no CSTART is specified, then the entire record is encrypted or decrypted. If CSTART is specified and CCOUNT is not specified, then the remainder of the record is encrypted or decrypted. The value of n must be a positive integer.
CCOUNT=n*	Optional parameter that specifies the number of columns to cipher. If the record is shorter than CSTART+CCOUNT, then the balance of the record after CSTART is ciphered. The value of n must be a positive integer.
COMPAT=4	Optional parameter that is a migration aid. Protegrity versions prior to 5.1 always specified that padding should be applied when data elements were created. This padding increased the length of the encrypted data by up to 16 characters. The z/OS products did not use this padding setting and instead used an encryption algorithm called CUSP that did not increase the size of the encrypted data. In addition, the z/OS products did not honor the Checksum and IV settings of the data element. With the capability of specifying the CUSP algorithms in the version 5.1 ESA, the z/OS products were enhanced to honor the padding, Checksum, or IV settings. To decrypt data encrypted by prior versions, it is IMPERATIVE to specify COMPAT=4. If you do not specify COMPAT=4, then decryption does not work, resulting in error codes. If you want the ciphering behavior for version 5.1 to be the same as in versions prior to 5.1, then you can also specify this. However, if you do not want the encrypted data to be larger than the clear text data, you should select CUSP methods when you define data elements. This is particularly true when the encrypted data may go into a PDS or fixed length PDSE.
EXTERNALIV=startcolumn	Specifies the starting column for the external InitializationValue to be used. If not specified, then no external IV is used.

Parameter	Description
	Startcolumn must be a positive numeric.
EIVCOUNT=numberofcolumns	Specifies how many columns should be included in the external IV. If EXTERNALIV is not specified, then do not specify this value. There is no default value and this value must be specified if EXTERNALIV is used. Numberofcolumns must be a positive numeric value.

Note: Decimal tokens are not supported since the File Protector does not return the required value when the data is detokenized.

* Non-length preserving tokens are not supported with CFU.

* Only CUSP algorithms and length-preserving tokens are valid for CSTART and CCOUNT parameters. Any other algorithm, including CUSP with KeyID or CRC, has the potential to increase the size of the protected data and would exceed the size chosen to be protected.

For more details on CUSP algorithms, refer to the section *CUSP* in the *Protection Methods Reference Guide 9.2.0.0.*

Note: While using externalIV, consider the following factors:

- Specify CSTART and CCOUNT along with EXTERNALIV and EIVCOUNT parameters
- EXTERNALIV value should not be in the range of column value specified in CSTART and CCOUNT
- Protection of the full file using EXTERNALIV is not possible.
- The external IV is ignored during encryption

6.2.1.2 Input File Specifications

This section mentions about the input file specifications.

These are the specifications for the input file:

- The input file is pointed to *DDNAME* with *PTYUT1*.
- It must be a standard z/OS sequential file and may be a UNIX System Services (USS) file.
- The sequential file may be a member of a partitioned data set.

Note: For USS files, z/OS does not maintain logical record lengths, so the *LRECL* or the *BLKSIZE* needs to be specified through JCL to allow the buffers to be the proper size to handle the longest record in the file. Tokens are not supported while performing protect or unprotect operations on USS files.

6.2.1.3 Output File Specifications

This section mentions about the input file specifications.

These are the specifications for the output file:

- The output file is pointed to *DDNAME* with *PTYUT2*.
- It must be a standard z/OS sequential file and may be a UNIX System Services (USS) file.

- The sequential file may be a member of a partitioned data set. If it is, then an ISPF style ENQ is done on the member name and the dataset name to get exclusive access to the member.

> **Note:** For USS files, z/OS does not maintain logical record lengths so the *LRECL* or the *BLKSIZE* needs to be specified through JCL to allow the buffers to be the proper size to handle the longest record in the file.

> **Note:** If the *RECFM, LRECL*, and/or *BLKSIZE* are not specified, then these are copied from the input file specifications.

6.3 Cryptographic Subsystem for z/OS

Cryptographic Subsystem for z/OS is a complex solution for protecting and unprotecting of QSAM/BSAM supported datasets.

Cryptographic Subsystem for z/OS consists of two main components, which are Subsystem Interface Access Method (SIAM) and Protegrity Crypto Access Method working as plugins to SIAM Subsystem (*PTYPAMLT* and *PTYPAMEX* modules) and these handle the actual file protection. *PTYPAMLT* module supports only length-preserving encryption algorithms, such as CUSP and length preserving tokens.

The SIAM subsystem intercepts QSAM or BSAM calls in your application programs using an MVS subsystem interface (SSI). The SSI can be installed dynamically or permanently. If installed permanently, then you must start IPL to complete the SSI installation. Updates to your SYS1.PARMLIB data set are required to install the subsystem permanently.

After installing the subsystem, whether dynamically or permanently, not all QSAM/BSAM requests are intercepted and processed by CSS. This only occurs when the required JCL changes are made.

Protegrity z/OS Cryptographic Access Method has the following characteristics:

- QSAM and BSAM access methods are supported.
- Supported AM macros are GET, PUT, READ, and WRITE.
- Fixed, variable, and undefined length record formats.
- Implemented as modules for Protegrity z/OS SIAM Subsystem.
- SUBSYS parameter of the DD statement requests encryption or decryption.
- Up to 16 fields can be encrypted with the same or different data elements. The field can be the whole record.
- Existing Protegrity z/OS infrastructure utilized.
- Only JCL changes are required and no changes are needed for the application code.

6.3.1 CSS Subsystem Usage

This section describes about CSS Subsystem usage.

To allow CSS to encrypt or decrypt data, you need to change the JCL DD statement for the required dataset. To allow QSAM/BSAM calls to be intercepted by CSS, update the DD statement for dataset as follows:

Before:

```
//APPDATA DD DSN=APP.DATA, DISP=SHR
```

After:

```
//APPDATA  DD SUBSYS=(SIAM,module-name[,dd-name][,D1][,C1][,L1] … ),
         DCB=(APP.DATA.ENCRYPTD[,BUFNO=1])
//APPDATA@ DD DSN=APP.DATA.ENCRYPTD,DISP=...
```

The following table explains the parameter values for the SUBSYS parameters.

Table 6-2: Parameter values for the SUBSYS parameters

Parameter	Description
1st parameter, module name	Module name which processes the QSAM or BSAM calls. Protegrity supplies PTYPAMLT and PTYPAMEX.
2nd parameter, optional	DD-name for encrypted file. For more information, refer to Point 1 in the Note mentioned below.
Dx, optional	Data element name for record field. Up to 16 fields can be defined.
Cx, optional	Field start column.
Lx, optional	Field length.

Note:

- If dd-name is omitted, then the default DD name is generated by adding "@" to the end of SUBSYS DD name, or by replacing the first character of SUBSYS DD name to "@" if it is 8 bytes long.
- If Dx is omitted, then the last one specified remains in effect.
- If Cx is omitted, then it defaults to the next byte after the previous field definition, or to one for the first field.
- If Lx is omitted, then it defaults to the remainder of record.
- You need to specify Cx in ascending order and avoid fields overlapping.
- If none of the fields are specified, then copy operation takes place.
- *PTYPAMEX* ignores columns specification for input (during decrypt processing) and retrieves it directly from the encrypted file.
- You may choose to execute a partial decrypt with *PTYPAMLT* by specifying other columns in the input.
- You may choose to permanently store definitions of various columns in the library and INCLUDE them into JCL, as below:

```
//LAYOUTS JCLLIB=(....)
// INCLUDE L1
…
//INPUT DD SUBSYS=(SIAM,PTYMAMLT,FILEDD,&L1)
//FILEDD DD DSN=...
where L1 member contains // SET L1='AES128C,1,16,,32,12' for example.
```

- Neither RECFM=VS or RECFM=VBS dataset is supported for input.
- There is no default DCB information for the SUBSYS dataset.
- Using CSS for decryption requires DCB parameters specified for the input file (encryption file) SUBSYS DD name. This may be coded explicitly or referenced to the existing input file (encryption file) as shown in the following examples:

```
//APPDATA DD SUBSYS=(SIAM,PTYPAMLT,DATAOUT,&FIELD1.&FIELD2),
// DCB=&PTYHLQ..FILEAMLT.DATA.RECFMF.ENC
//DATAOUT DD DISP=SHR,DSN=&PTYHLQ..FILEAMLT.DATA.RECFMF.ENC


OR


//APPDATA DD SUBSYS=(SIAM,PTYPAMLT,,&FIELD1.&FIELD2),
// DCB=&PTYHLQ..FILEAMLT.DATA.RECFMF.ENC
//APPDATA@ DD DISP=SHR,DSN=&PTYHLQ..FILEAMLT.DATA.RECFMF.ENC
```

The DCB parameters in the output file (decryption file) DD name is optional.

Therefore, if the application program does not provide such defaults, then RECFM, LRECL, or BLKSIZE should be given from the SUBSYS DD. These may be coded explicitly or obtained by the system when DCB sub-parameter refers to the existing dataset. For example, DCB=MODEL.DATASET.NAME. BUFNO=1 is recommended instead of default 5 since there is no actual I/O for the SUBSYS dataset.

By default, the following parameters are copied from the application DCB to the DCB for a file at SUBSYS DCB OPEN time by the *PTYPAMLT*:

- RECFM - all record formats

- LRECL - F/V records
- BLKSIZE - U records

This allows for MVS to choose *BLKSIZE* automatically for a new output dataset or process input *BLKSIZE* other than what the application program expects for the F/V records.

If you specify DCB=INTVL=1 in the DD statement for a file, then BLKSIZE is also copied from application DCB (of course, it is either specified by application program, or inherited from SUBSYS DD). *PTYPAMEX* uses its own format of encrypted file. You can view a CSS sample code in the section *JCL Sample Code for Using CSS*.

6.4 CSS and CFU Comparative Properties

This section gives suggestions and comparative information about CSS and CFU. The scenarios where CSS or CFU might be used are also explained. The advantages of using *PTYPAMEX* for CSS and *PTYPAMLT* for CSS are covered in this section.

CFU is a file utility that takes an input file and manipulates it to create an output file. The entire file is processed. You can specify what data records and what columns within those records to manipulate; however, the entire input file is written to the output file unless an error occurs. This means if you use CFU with a protected master file to generate a new protected master file, then you need two intermediate clear-text files in between: protected master to clear-text temporary master to processing to generate clear-text next temporary master to new protected master. Also, CFU has a limitation that only one set of columns or the entire data record can be manipulated, whether you encrypt or tokenize.

CSS utilizes intercepts of standard z/OS input or output processing. The CSS routines intercept the z/OS input or output buffer and manipulate the data before the application views it (for input), or after the application writes to the output buffer and before z/OS puts the data into the disk buffer (for output). Both can be done at the same time and multiple sets of data columns can be manipulated during this process using the same or different data elements. So you can, for example, use encryption for credit card numbers and use tokens for SSN values or dates of birth. After CSS is installed, you can add protection to applications by changing the JCL to add the CSS intercepts to the required files. No temporary clear-text files are necessary; protected master file intercepted on input to application to intercepted new master file as written by the application. For more information, refer to the example in the DEMOCSS JCL in the `PTYxxxx.FPSAMP` library.

Table 6-3: Property Comparison – CSS and CFU

Properties	PTYPAMLT	PTYPAMEX	PTYPFILE
Encrypted file format	Dictated by application	Special (always VB with header records)	Dictated by application
Algorithms supported	CUSP and LP tokens	Any supported by z/OS DPS	CUSP and LP tokens
Max record length	Up to z/OS supported length	Less than the length supported by the OS	Up to z/OS supported length
Fields specification	Should be the same for the protect / unprotect operations	Can be specified only for the protect operations	Should be the same for the protect / unprotect operations
Concatenated input	Supported	Not supported	Supported
Number of protected fields	Up to 16 non-adjacent	Up to 16 non-adjacent	1
Input file verification	N/A	Yes	N/A

Chapter 7

Database Solutions on Mainframe z/OS

The DB2 Database Protector for z/OS is a set of products that provides for encrypting and decrypting DB2 data. You can encrypt or decrypt individual columns using FPR (FIELDPROC) or the entire row using EPR (EDITPROC).

The products use standard DB2 exits to invoke the encryption or decryption. These exits and their usage are documented in manuals from IBM (refer to the IBM website for downloads): *DB2 SQL Reference and DB2 Administration Guide*. Certain restrictions from those manuals are documented in this manual, but we recommend you refer to those documents for additional information.

> **Note:** The IBM software product, z/OS Integrated Cryptographic Service Facility (ICSF), is not required by Protegrity DPS. It is not required to use any of the supported cryptographic hardware and it is not required to perform software only encryption and decryption.

Protegrity Database Protector for z/OS consists of several modules:

- Cryptographic Services Manager (*PTYPCSM* module and its associated modules)
- The PEP Server executable and its pieces which run under Open MVS (UNIX System Services or USS)
- The DB2 FIELDPROC function (*PTYPFPR* module)
- The DB2 EDITPROC function (*PTYPEPR* module).
- The DB2 UDFs

The DB2 SIGNON exit routine can be used for CICS and IMS environments.The sample code for DSN3SSGN with Protegrity specific changes (call to *PTYPSGN*) is included in `PTYxxxx.DB2SAMP` library. If the DB2 SIGNON exit is not used, then the DB2 Authorized User is used.

DB2 Database Protectors for z/OS support user access through multiple secondary authorization IDs (RACF group name). DB2 FIELDPROC and EDITPROC routines grant access to a user defined in a policy created using ESA by matching the Policy role user with the DB2 User ID. If match is not found, then the Policy role user will try to match with secondary authorization ID which can be set using the DB2AUTHIDLVL parameter in the parameter file for Cryptographic Services Manager. The DB2AUTHIDLVL parameter identifies the number of secondary User IDs, DB2 FIELDPROC and EDITPROC routines should try to match before the error "User Access Denied" is displayed. For more information about setting parameter, refer to the section *Installing the z/OS Product*.

Only one of the FIELDPROC or EDITPROC functions is necessary along with the other services to run the Database Protector. Both FIELDPROC and EDITPROC may be used at the same site, but not on the same table. Sample jobs and table definitions are also included in the product.

Before you install the DB2 portions of the product, you should install and customize the Protegrity for z/OS Cryptographic Servers. For more information about installing the Cryptographic Servers, refer to the section *Installing the z/OS Product*.

7.1 Installing and Configuring

This section explains the installation and configuration of the Database Protector solutions on Mainframe z/OS.

The following sub-sections explain the installation of Database Protector Solution on Mainframe z/OS. It provides a high-level setup overview as well as post-installation verification steps.

- *Database Protector Setup*
  - *DB2 Protector Installation*
  - *DB2 Protector Installation Verification*

7.1.1 Setting up Database Protector on Mainframe z/OS

The following procedure explains in high level how to install DB2 Database Protector on Mainframe z/OS.

➤ To install Database Protector on Mainframe z/OS:

1. Install PEP Server. For more information about installing the PEP server, refer to the section *Installing the PEP Server*.
2. Install Protegrity Cryptographic Services. For more information about installing the Cryptographic Services, refer to the section *Installing the Cryptographic Server*.
3. Install the DB2 Database Protector. For more information about installing the Database Protector, refer to the section *DB2 Protector Installation*.
4. Activate Protegrity Data Security Platform. For more information about activating the Protegrity Data Security Platform, refer to the section *Activating Protegrity Data Security Platform*.
5. Deploy a test policy. For more information about deploying a test policy, refer to the section *Deploying a Test Policy*.
6. Verify the PEP server installation. For more information about verifying the installation of the PEP server, refer to the section *Verifying the Installation*.
7. Verify the protector installation. For more information about verifying the installation of the Database Protector on z/OS, refer to the section *Protector Installation Verification*.

7.1.1.1 DB2 Protector Installation

This section outlines the required steps to install and customize the database protector for z/OS.

➤ To install the DB2 Database Protector for z/OS:

1. The files `PTYxxxx.DB2AUTH`, `PTYxxxx.DB2LOAD`, and `PTYxxxx.DB2SAMP` are restored using the `LODxxxD.jcl`. For more information, refer to the section *Prerequisites for Installing the z/OS Product*.

`DB2AUTH`	Load module library, needs to be PDS
`DB2LOAD`	Load module library

DB2SAMP	RECFM F LRECL 80 JCL type library, needs to be PDS

> **Note:** It is recommended to install the IBM z/OS hotfix PI32471 for DB2 before using the FIELDPROC module with non-CUSP algorithms.

Under DB2 rules, *DB2AUTH* must be APF authorized, or the load modules included must be copied to authorized libraries. If the modules are copied, then there are aliases in DB2AUTH that should be maintained.

The default protection or unprotection behavior is for EDITPROC or FIELDPROC to attempt the hardware encryption or decryption, and fall back to software when the hardware is not available. Aliases allow the site to change that behavior. The aliases do not necessarily need to be maintained if the modules are copied:

- *PTYPFPS* is an alias of *PTYPFPR*. This is the software only alias for FIELDPROC. It forces the FIELDPROC module to use software encryption and becomes a performance issue when hardware encryption is available.
- *PTYPEPS* is an alias of *PTYPEPR*. This is the software only alias for EDITPROC. It forces the EDITPROC module to use software encryption and becomes a performance issue when hardware encryption is available.

2. Add *DB2AUTH* (or the copied-to library) to the DB2 Database Facility start-up procedure (for example, *DB8GDBM1*). DB2 needs to be restarted prior to attempting to define a FIELDPROC or EDITPROC.

7.1.1.2 Protector Installation Verification

This section demonstrates the installation verification of Database Protector on z/OS.

Restart DB2 if it was not already restarted during installation. Start the *PTYCSRV* procedure to start the Protegrity Cryptographic Services Manager Task. This starts the other servers. Use the procedure customized during installation. All services can be stopped by stopping the *PTYCSRV* procedure from an operator console.

The following sections verifies the installation:

- *To verify FIELDPROC*
- *To verify EDITPROC*

7.1.1.2.1 Verifying FIELDPROC

These steps explain the installation verification of FIELDPROC.

➤ To verify FIELDPROC Installation:

1. Run sample SQL script *PTYxxxx.DB2SAMP*(FPRTEST) using SPUFI.
2. Stop the test table spaces: STOP DB(DSNDB04) SPACENAME(FPR*).
3. Run the job *PTYxxxx.DB2SAMP*(FPRPRNT) to print the table spaces generated by FPRTEST.
4. Examine the FPRPRNT output to verify that the CCN and LNAME columns are shown in clear text for the FPRTEST and FPRTEST2 tables and in cipher text for the FPRTEST1 table.

7.1.1.2.2 Verifying EDITPROC

These steps explain the installation verification of EDITPROC.

➤ To verify EDITPROC Installation:

1. Submit the job *PTYxxxx.DB2SAMP*(PTYETEST) to generate the sample test EDITPROC.
2. Run sample SQL script *PTYxxxx.DB2SAMP*(EPRTEST) using SPUFI.
3. Stop the test table spaces: STOP DB(DSNDB04) SPACENAME(EPR*).
4. Run the job *PTYxxxx.DB2SAMP*(EPRPRNT) to print the table spaces generated by EPRTEST.
5. Examine the EPRPRNT output to verify that the row data are shown in clear text for the EPRTEST and EPRTEST2 tables and in cipher text for the EPRTEST1 table

7.2 FIELDPROC Usage

DB2 provides a column routine (DB2 exit) called a field procedure (FIELDPROC) which is used to encode and decode customer short-string data stored in a single column in SQL tables.

DB2 calls the field procedure during LOAD, INSERT, UPDATE, and SELECT. The procedure is also called during data comparison on the column unless the comparison is for LIKE. If there are any other edit routines (for example EDITPROC) invoked for data encoding, then FIELDPROC is invoked first. For data decoding, FIELDPROC is invoked last. The column data on which the operation is being performed is passed to the program (written in assembler) identified in the FIELDPROC and the contents of the column may change in this program.

For full details of FIELDPROC operations, refer to the *IBM DB2 Administration Guide*.

The Protegrity field procedure, *PTYPFPR*, is invoked to encrypt or decrypt the fields for which the FIELDPROC clause is specified on the CREATE TABLE or ALTER TABLE SQL statement, for example:

```
CREATE TABLE CUSTTBL (
        ...
        CCN CHAR(16) FIELDPROC PTYPFPR (‘policy data element name’
        [, ‘access denied substitute value’])
        ...
```

The FIELDPROC clause can be specified only on short string columns (such as: CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, NOT LONG VARCHAR, INT, and so on). You can use IBM DB2 Tools, CA Platinum, BMC Change Manager, or a similar product to view a table and its properties. This way you can tell that an FIELDPROC exists for a column within a table.

The Protegrity Database Protector for z/OS FIELDPROC program *PTYPFPR* takes zero, one, or two parameters. With no parameter specified for *PTYPFPR*, the hardcoded key that was used in previous versions (prior to version 4.3.0.0) of Protegrity Database Protector for z/OS is used. Using *PTYPFPR* this method does NOT enforce the Protegrity security policy. Data is encrypted and decrypted as *PTYPFPR* is called from DB2 without regard for users or time restrictions. Only DB2 access restrictions limit data usage.

To use the Protegrity security policy for data protection, you must specify at least the first parameter. The policy restrictions are enforced in addition to any DB2 restrictions. A specific user can be allowed access to a TABLE via DB2, and disallowed access to a specific column via the security policy. However, denying DB2 access to a TABLE and allowing access via the security policy still denies access to the user since the policy enforcement is in addition to the DB2 enforcement, not instead of the DB2 enforcement. For example:

```
CREATE TABLE CUSTTBL (
       ...
       CCN CHAR(16) FIELDPROC PTYPFPR
       ...
```

The first parameter specifies the policy data element name that should be used to enforce the policy for this column. FIELDPROC *PTYPFPR* can be specified for multiple columns in one or more tables with the same or different policy data element names. If the same policy data element name is used, then the same policy parameters (time, users, and so on) are enforced. The parameter value is specified as a quoted string to DB2. It must immediately follow the name of the FIELDPROC

PTYPFPR, and must be enclosed in parentheses. If the second parameter is used, then the closing parentheses (“)”) is after the second parameter and a comma follows the first parameter. For example:

```
CREATE TABLE CUSTTBL (
       ...
       CCN CHAR(16) FIELDPROC PTYPFPR (‘PTY_IVP_FPRTEST_CCN’)
       ...
```

The optional second parameter specifies what value to substitute for the column value in the SELECT case when policy access is denied, but DB2 access is allowed. If this parameter is NOT specified, and policy access is denied, then a DB2 error is generated by *PTYPFPR* when a SELECT attempt is made by a user that does not have policy permission to access the data. The substitute value is specified as a quoted string. When an attempt is made for update (including INSERT, CHANGE, or DELETE) by a user that does not have policy access, but does have DB2 access, then a DB2 error gets generated. For example:

```
CREATE TABLE CUSTTBL (
       ...
       CCN CHAR(16)
           FIELDPROC PTYPFPR (‘PTY_IVP_FPRTEST_CCN’, ‘*****’)
       LNAME VARCHAR(14)
             FIELDPROC PTYPFPR (‘PTY_IVP_FPRTEST_LNAME’, ‘#######’)
   ...
```

7.2.1 DB2 Passes Control to Field Procedure

DB2 has the following specifics when the control is passed to a Field Procedure.

- ***CREATE TABLE tablename …COLUMN columnname FIELDPROC fieldprocname …*** prompts DB2 to load and validate FIELDPROC.
- FIELDPROC runs as an extension of DB2 with all of DB2’s privileges
- Before the FIELDPROC is added to the table, the data needs to be unloaded. Then the FIELDPROC is added, and the data re-loaded. This ensures that existing data is encrypted in the table. Other methods than unload or reload may be used to get the table data encrypted including ALTER TABLE.

> **Note:** DB2 does not pass control if mass delete is requested.

7.2.2 FIELDPROC Restrictions and Limitations

DB2 has restrictions and limitations on the use of a Field Procedure. These are documented in the following manuals from IBM: *DB2 SQL Reference* and *DB2 Administration Guide*.

The FIELDPROC restrictions and limitations are documented below, but these may not be the only restrictions.

- A FIELDPROC name must be unique and cannot be the same as an existing DB2 entity name.
- The column must not be a security label.
- The column must not be defined with DEFAULT.
- A field procedure is not invoked for null values. A user could insert null values regardless security policy access constraints.
- A field procedure is not invoked for a DELETE operation without a WHERE clause on a table in a segmented table space.
- Only short-string datatypes are supported: CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, etc.
- DB2 FIELDPROC currently support INT and SMALLINT. For using Token Type Integer with Size 2, create DB2 column with CHAR(2) and for using Token Type Integer with Size 4, create DB2 column with CHAR(4), otherwise it returns an error with a message “Invalid Input Size”.
- DB2 does not have support in FIELDPROC or UDFs for BIGINT. Using an INTEGER token with size of 8 returns an error.
- Predicates that do comparisons only work for equals or not equals. Range predicates, less than, and greater than predicates do not work correctly.

- Left justified data for integer tokens and non-numeric characters are not supported with FIELDPROC. Attempt to use of these will show error.

7.2.3 Unprotect Only FIELDPROC

The Protegrity unprotect only field procedure, *PTYPFPU*, is invoked to decrypt the fields for which the FIELDPROC clause is specified.

The data can be protected using z/OS or any other protector on any platform. The data when inserted in columns on which unprotect only Fieldproc clause is specified, it will copy data as is and not perform any encryption to avoid double encryption. When the column with FIELDPROC PTYPFPU is selected, the data is decrypted using the policy data element name specified in column definition during CREATE TABLE.

> **Note:** For VARCHAR column, the column length should be increased by 2 bytes to handle the length.
>
> Non-length preserving tokens are supported with only VARCHAR column type.
>
> For non-z/OS Protectors,
>
> - The data protected using tokens, should be transferred to z/OS as text, to translate the data from ASCII to EBCDIC.
> - The data protected using encryption, should be transferred to z/OS as binary, to keep the data as is. CCSID ASCII should be used during CREATE TABLE.
>
> While using unprotect only field procedure, it is recommended to use tokens for data protection on non-z/OS protectors.
>
> The *PTYPFPU* does not support unprotecting a data that is protected with Unicode tokens.

Following is an example where the Protegrity unprotect only field procedure, *PTYPFPU*, is specified on the CREATE TABLE or ALTER TABLE SQL statement.

```
CREATE TABLE CUSTTBL (
 ...
 CCN CHAR(16) FIELDPROC PTYPFPU (‘policy data element name’
[, ‘access denied substitute value’])
 ...
```

For details on FIELDPROC operations, refer to the *IBM DB2 Administration Guide*.

7.3 EDITPROC Usage

This section explains the EDITPROC usage .

DB2 provides an Edit Procedure (DB2 exit) called an EDITPROC which is used to encode and decode customer data stored in SQL tables. As such, many IBM customers have previously used this exit to compress data and recently to encrypt and decrypt their data and are generally aware of the exit’s existence. In addition, IBM and some vendors utilize this exit in their encryption solution for DB2 on mainframes. For more details of EDITPROC operations, refer to the *IBM DB2 Administration Guide*.

DB2 calls the edit routine exit during LOAD, INSERT, and UPDATE. The row on which the operation is being performed is passed to the program (written in assembler) identified in the EDITPROC and the contents of the row may change in this program. You can use IBM DB2 Tools, CA Platinum, BMC Change Manager, or a similar product to view a table and its properties. This way, you can tell that an EDITPROC exists for a table.

Since DB2 does not allow user parameters to be passed to EDITPROC, the site must generate a unique EDITPROC for each unique policy data element to be used for row level encryption or decryption. The policy data element name is included in the

customized EDITPROC. A sample job (*PTYETEST*) is provided in the *PTYxxxx.DB2SAMP* library to generate the unique EDITPROC. The sample job is set to generate the EDITPROC for the examples shown below.

An EDITPROC is added to a table using the CREATE TABLE DML. The Protegrity field procedure, *PTYPEPR*, is invoked to encrypt or decrypt the row for which the EDITPROC clause is specified on the CREATE TABLE SQL statement.

```
CREATE TABLE PTY.EPRTEST1 … EDITPROC PTYPEPR;
```

7.3.1 DB2 Passes Control to an Edit Procedure

DB2 has the following specifics when the control is passed to an Edit Procedure.

- ***CREATE TABLE tablename …COLUMN columnname EDITPROC editprocname …*** adds EDITPROC.
- EDITPROC runs as an extension of DB2 with all of DB2's privileges.
- Before the EDITPROC is added to the table, the data needs to be unloaded. Then the EDITPROC is added, and the data re-loaded. This ensures that existing data is encrypted in the table. Other methods than unload or reload may be used to get the table data encrypted.

7.3.2 EDITPROC Restrictions and Limitations

DB2 has restrictions and limitations on the use of an Edit Procedure.

Following are the EDITPROC restrictions and limitations:

- An EDITPROC name must be unique and cannot be the same as an existing DB2 entity name.
- When using an EDITPROC, the tablespace page size is reduced by 10 bytes.
- The table must not contain a LOB or ROWID column.
- Indexes are not encrypted.
- Only CUSP encryption logics are supported with EDITPROC.
- EDITPROC does not support masking

7.3.3 EDITPROC with Compression and Decompression support

DB2 Edit Procedure (EDITPROC) allows compression and decompression of customer data.

The customer site can choose to compress the data, in addition to encryption, while using Protegrity's EDITPROC routine. During LOAD, INSERT, and UPDATE, the data will be compressed first, using the compression routine provided by the customer site, and then the compressed data would be encrypted. During UNLOAD and SELECT, the data will be decrypted first, and then the decrypted data will be decompressed using the compression routine provided by the customer site.

To compress the data using EDITPROC routine, the site must configure the parameter "COMPRESS=Y" in the customized EDITPROC routine that is generated with the PTYETEST sample job. The default value is "COMPRESS=N".

Note:

- The sample compression routine (*PTYETEST*) provided by Protegrity is based on the sample EDITPROC compression routine DSN8HUFF (HUFFMAN COMPRESSION EDITPROC), which is provided by IBM.
- The sample routine (DSN8HUFF) that compresses the Huffman data is provided in the *prefix.SDSNSAMP* library.
- The customer site can provide their own compression routine, which should be named as "*PTYPCMPR*". This customized compression routine must follow the IBM sample for passing a parameter.

7.4 Installing Protegrity User Defined Functions

This section describes all the Protegrity UDFs that are available for Mainframe z/OS.

The following sub-sections explain the installation of Protegrity UDFs on Mainframe z/OS:

- *Pre-Installation Configuration*
- *UDFs Installation*

7.4.1 Pre-Installation Configuration

This section describes the steps that you need to take before installing the specific DB2 UDFs. It includes some Work Load management configuration and user permissions in the DB2 subsystem.

To configure the DB2 Database Protector:

1. Create the appropriate WLM (Work Load Management) application environment, considering the target libraries in the *STEPLIB* part of the Work Load Management JCL (Job Control Language) and “Limit on starting server address space for subsystem instance”.

   Sample Work Load Management configuration:

   ```
   Action Application Environment Name Description
   __ PROTGRIT Stored procedures for
   Protegrity
   Appl Environment Name . . PROTGRIT
   Description . . . . . . . Stored procedures for Protegrity
   Subsystem type . . . . . DB2
   Procedure name . . . . . TSNWLM
   Start parameters . . . . APPLENV=PROTGRIT,DB2SSN=db2ssid,NUMTCB=1
   Limit on starting server address spaces for a subsystem instance:
   No limit
   ```

   The JCL procedure that is used in TSNWLM:

   ```
   //*************************************************************
   //* WLM-ESTABLISHED STORED PROCEDURES ADDRESS SPACE *
   //*************************************************************
   //TSNWLM PROC RGN=0M,APPLENV=PROTGRIT,DB2SSN=db2ssid,NUMTCB=1
   //IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
   // PARM='&DB2SSN,&NUMTCB,&APPLENV'
   //STEPLIB DD DISP=SHR,DSN=DSNB10.RUNLIB.LOAD
   // DD DISP=SHR,DSN=CEE.SCEERUN
   // DD DISP=SHR,DSN=DSNB10.SDSNLOAD
   // DD DISP=SHR,DSN=hlqual.UDFUDFX <== Protegrity UDF procedure library
   //SYSOUT DD SYSOUT=D,DEST=SPF
   //SYSPRINT DD SYSOUT=D,DEST=SPF
   //SYSUDUMP DD SYSOUT=D,DEST=SPF
   //SYSABEND DD SYSOUT=D,DEST=SPF
   ```

   > **Note:** It is recommended that the value of ***REGION*** parameter should be at least 512M.

2. Create a user who runs the scripts to create the UDFs. The user must have these grants in the DB2 database:
   - GRANT USE OF STOGROUP sgname TO PTY
   - GRANT CREATEDBA TO PTY
   - GRANT BINDADD TO PTY
   - GRANT PACKADM ON COLLECTION PTY TO PTY WITH GRANT OPTION
3. Grant access to UDFs for users. Users need to have permission to access the Protegrity User Defined Functions. The user must have this grant in the DB2 database:
   - GRANT EXECUTE ON FUNCTION PTY.* TO user

where user is the user or group that you want to have authorized to use the UDFs. You may need GROUP specified in front of the group name

7.4.2 Installing UDFs

This section describes how to create and install UDFs for a DB2 database. Follow the below procedure to install the UDFs:

To install UDFs:

1. Files *PTYxxxx.UDFSQL*, *PTYxxxx.UDFUDFX*, and *PTYxxxx.UDFSAMP* are restored using *LODxxxD.jcl*.

UDFUDFX	contains executables of UDFs
UDFSQL	contains SQL procedures to create and delete UDFs
UDFSAMP	contains the same invocations of a UDF

   UDFUDFX is a PDS-E. The other two files are PDS files.

2. Edit *PTYxxxx.UDFSQL* (CREATOBJ) member and change WLM ENVIRONMENT in each CREATE FUNCTION query to the Workload Manager Environment that you created above (APPLENV=PROTGRIT).
3. Include *PTYxxxx.UDFUDFX* in the STEPLIB concatenation for the DB2 Workload Manager start-up procedure.
4. Connect as a privileged user and open the script *PTYxxxx.UDFSQL* (CREATOBJ), where *PTYxxxx* is the high level qualified used to load the library.
5. Execute *PTYxxxx.UDFSQL* (CREATOBJ).
6. Install the PEP Server as it is described in the section *PEP Server Installation*. Note that this is the only z/OS server that you need to install if you are only using the UDFs. You can use the same PEP Server for UDFs and FIELDPROC or EDITPROC.
7. Build and deploy a policy as is described in previous sections. You may use the same data element names as described above or use different data element names.

7.4.3 General UDFs

This section lists the general UDFs including syntax.

7.4.3.1 pty.whoami

This function returns the name of the user who is currently logged in.

The external name is PDWHOAMI.

pty.whoami()

Parameters

None

Returns

This UDF returns the name of user logged in to the database as VARCHAR(256).

Example

```
SELECT PTY.whoami() from SYSIBM.SYSDUMMY1;
```

7.4.3.2 pty.getversion

This function returns the version of the product.

The external name is PDGETVER.

pty.getversion()

Parameters

None

Returns

This UDF returns the version of the product as VARCHAR(256).

Example

```
SELECT PTY.getversion() from SYSIBM.SYSDUMMY1;
```

7.4.3.3 pty.getcurrentkeyid

This function returns the current key ID for a data element. It is typically used together with **getkeyid** to determine if some data is protected with the most recent key for a given data element.

The external name is PDGCKEY.

pty.getcurrentkeyid(communicationid INTEGER, dataelement VARCHAR)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies location where policy can be found. Must be the same as configured in `pepserver.cfg`
dataelement	VARCHAR(64)	Specifies the name of data element.

Returns

This UDF returns the current key ID as INTEGER.

Example

```
SELECT PTY.getcurrentkeyid(8,'data-element name') from SYSIBM.SYSDUMMY1;
```

7.4.3.4 pty.getkeyid

This function returns the current key ID that was used to protect an item of data. It is typically used together with getcurrentkeyid to determine if some data is protected with the most recent key for a given data element.

The external name is PDGKEY.

pty.getkeyid(communicationid INTEGER, dataelement VARCHAR, data VARCHAR FOR BIT DATA)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies location where policy can be found. Must be the same as configured in `pepserver.cfg`
dataelement	VARCHAR(64)	Specifies the name of data element.
data	VARCHAR(32672) FOR BIT DATA	Specifies the data that has been encrypted using a key ID.

Returns

This UDF returns the key ID as INTEGER.

Example

```
SELECT PTY.getkeyid(8,'data-element name',pty.ins_enc_varchar (8,'data-element name' ,
'abc123456',0) ) from SYSIBM.SYSDUMMY1;
```

7.4.4 Access Check UDFs

These UDFs can be used to check access permissions. The procedures will pass if user has access, otherwise it will cast an exception with the reason for failure.

7.4.4.1 pty.have_sel_perm

This function is used to determine if the user has select access to a data element.

The external name is PDHSPERM.

pty.have_sel_perm(communicationid INTEGER, dataelement VARCHAR)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies location where policy can be found. Must be the same as configured in *pepserver.cfg*.
dataelement	VARCHAR(64)	Specifies the name of data element.

Returns

If the user has select access, the UDF returns an INTEGER value (1).

> **Note:** If the user does not have the select access, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.have_sel_perm(8,'data-element name') FROM SYSIBM.SYSDUMMY1;
```

7.4.4.2 pty.have_upd_perm

This function is used to determine if the user has update access to a data element.

The external name is PDHUPERM.

pty.have_upd_perm(communicationid INTEGER, dataelement VARCHAR)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies location where policy can be found. Must be the same as configured in *pepserver.cfg*.
dataelement	VARCHAR(64)	Specifies the name of data element.

Returns

If the user has update access, the UDF returns an INTEGER value (1).

> **Note:** If the user does not have the update access, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.have_upd_perm(8,'data-element name') FROM SYSIBM.SYSDUMMY1;
```

7.4.4.3 pty.have_ins_perm

This function is used to determine if the user has insert access to a data element.

The external name is PDHIPERM.

pty.have_ins_perm(communicationid INTEGER, dataelement VARCHAR)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.

Returns

If the user has insert access, the UDF returns an INTEGER value (1).

> **Note:** If the user does not have the insert access, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.have_ins_perm(8,'data-element name') FROM SYSIBM.SYSDUMMY1;
```

7.4.4.4 pty.have_del_perm

This function is used to determine if the user has delete access to a data element.

The external name is PDHDPERM.

pty.have_del_perm(communicationid INTEGER, dataelement VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

If the user has delete access, the UDF returns an INTEGER value (1).

> **Note:** If the user does not have the delete access, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.have_del_perm(8,'data-element name',0) FROM SYSIBM.SYSDUMMY1;
```

7.4.4.5 pty.del_check

This function is used to determine if the user has delete access to a data element.

The external name is PDDCHECK.

pty.del_check(communicationid INTEGER, dataelement VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

If the user has delete access, the UDF returns an INTEGER value (1).

> **Note:** If the user does not have the delete access, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty. del_check(8,'data-element name',0) FROM SYSIBM.SYSDUMMY1;
```

7.4.5 VARCHAR UDFs

These UDFs can be used to encrypt or decrypt VARCHAR data.

> **Note:** z/OS UDFs do not support LONG VARCHAR data type.

7.4.5.1 pty.ins_enc_varchar

This function is used to encrypt data with a data element.

The external name is PDIVCHR.

pty.ins_enc_varchar(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672)	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(32672) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (CCN VARCHAR(20) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (CCN)
VALUES (PTY.ins_enc_varchar(8,'data-element','4234567890123456',0));
```

7.4.5.2 pty.upd_enc_varchar

This function is used to encrypt data with a data element.

The external name is PDUVCHR.

pty.upd_enc_varchar(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672)	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(32672) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (CCN) =
PTY.upd_enc_varchar(8,'data-element','4234567890123456',0);
```

7.4.5.3 pty.sel_dec_varchar

This function is used to decrypt data with a data element.

The external name is PDSVCHR.

pty.sel_dec_varchar(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element
input_data	VARCHAR(32672) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as VARCHAR(32672).

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_varchar(8,'data-element',CCN,0) as CCN
from table-name;
```

7.4.5.4 pty.ins_varchar

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDIVCHRM.

pty.ins_varchar(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.

Name	Type	Description
input_data	VARCHAR(32672)	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(32672).

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (CCN VARCHAR(20))
CCSID EBCDIC IN DATABASE DSNDB04;

INSERT into table-name (CCN)
VALUES (PTY.ins_varchar(8,’token-element','4234567890123456',0));
```

7.4.5.5 pty.upd_varchar

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDUVCHRM.

pty.upd_varchar(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies location where policy can be found. Must be the same as configured in `pepserver.cfg`
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672)	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(32672).

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (CCN) =
PTY.upd_varchar(8,'token-element','4234567890123456',0);
```

7.4.5.6 pty.sel_varchar

This function is used for no encryption with a data element as well as for de-tokenization.

The external name is PDSVCHRM.

pty.sel_varchar(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672)	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as VARCHAR(32672).

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT (PTY.sel_varchar(8,'token-element',CCN,0) as CCN
   from table-name;
```

7.4.5.7 pty.ins_hash_varchar

This function uses the hash function to protect the data using a data element.

The external name is PDIVCHRH.

pty.ins_hash_varchar(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672)	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the hash value as VARCHAR(32672) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (CCN VARCHAR(20) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (CCN)
VALUES (pty.ins_hash_varchar(10,'HASHING','567#$%JEEVES',0));
```

7.4.5.8 pty.upd_hash_varchar

This function uses the hash function to protect the data using a data element.

The external name is PDUVCHRH.

pty.upd_hash_varchar(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672)	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the hash value as VARCHAR(32672) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (CCN) =
pty.upd_hash_varchar(10,'HASHING','HSHAIMN12345',0);
```

7.4.6 VARCHAR FOR BIT DATA UDFs

These UDFs can be used to encrypt or decrypt VARCHAR FOR BIT DATA.

7.4.6.1 pty.ins_enc_varcharfbd

This function is used to encrypt data using a data element.

The external name is PDIVCHRF.

pty.ins_enc_varcharfbd(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(32672) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (CCN VARCHAR(20) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (CCN)
VALUES (PTY.ins_enc_varcharfbd(8,'data-element','4234567890123456',0));
```

7.4.6.2 pty.upd_enc_varcharfbd

This function is used to encrypt data with a data element.

The external name is PDUVCHRF.

pty.upd_enc_varcharfbd(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(32672) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (CCN) =
PTY.upd_enc_varcharfbd(8,'data-element','4234567890123456',0);
```

7.4.6.3 pty.sel_dec_varcharfbd

This function is used to decrypt data with a data element.

The external name is PDSVCHRF.

pty.sel_dec_varcharfbd(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as VARCHAR(32672) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_varcharfbd(8,'data-element',CCN,0) as CCN
from table-name;
```

7.4.6.4 pty.ins_varcharfbd

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDIVCHF2

pty.ins_varcharfbd(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.

Name	Type	Description
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(32672) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (CCN VARCHAR(20) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (CCN)
VALUES (PTY.ins_varcharfbd(8,token-element','4234567890123456',0));
```

7.4.6.5 pty.upd_varcharfbd

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDUVCHF2.

pty.upd_varcharfbd(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(32672) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (CCN) =
PTY.upd_varcharfbd(8,'token-element','4234567890123456',0);
```

7.4.6.6 pty.sel_varcharfbd

This function is used for no encryption with a data element as well as for de-tokenization.

The external name is PDSVCHF2.

pty.sel_varcharfbd(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(32672) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as VARCHAR(32672) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_varcharfbd(8,'token-element',CCN,0) as CCN
from table-name;
```

7.4.7 CHAR UDFs

These UDFs can be used to encrypt and decrypt CHAR data.

7.4.7.1 pty.ins_enc_char

This function is used to encrypt data with a data element.

The external name is PDIVCHR.

pty.ins_enc_char(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(254)	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(300) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (CCN VARCHAR(20) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (CCN)
   VALUES (PTY.ins_enc_char(8,'data-element','4234567890123456',0));
```

7.4.7.2 pty.upd_enc_char

This function is used to encrypt data with a data element.

The external name is PDUVCHR.

pty.upd_enc_char(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(254)	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(300) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (CCN) =
   PTY.upd_enc_char(8,'data-element','4234567890123456',0);
```

7.4.7.3 pty.sel_dec_char

This function is used to decrypt data with a data element.

The external name is PDSVCHR.

pty.sel_dec_char(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR (300) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as VARCHAR(254).

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_char(8,'data-element',CCN,0) as CCN
from table-name;
```

7.4.8 CHAR FOR BIT DATA UDFs

These UDFs can be used to encrypt and decrypt CHAR FOR BIT DATA.

7.4.8.1 pty.ins_enc_charfbd

This function is used to encrypt data with a data element.

The external name is PDIVCHRF.

pty.ins_enc_charfbd(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(254) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(300) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (CCN VARCHAR(20) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (CCN)
VALUES (PTY.ins_enc_charfbd(8,'data-element','4234567890123456',0));
```

7.4.8.2 pty.upd_enc_charfbd

This function is used to encrypt data with a data element.

The external name is PDUVCHRF.

pty.upd_enc_charfbd(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(254) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(300) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (CCN) =
   PTY.upd_enc_charfbd(8,'data-element','4234567890123456',0);
```

7.4.8.3 pty.sel_dec_charfbd

This function is used to decrypt data with a data element.

The external name is PDSVCHRF.

pty.sel_dec_charfbd(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(300) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as VARCHAR(254) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_charfbd(8,'data-element',CCN,0) as CCN
from table-name;
```

7.4.9 DATE UDFs

These UDFs can be used to encrypt and decrypt DATE data.

7.4.9.1 pty.ins_enc_date

This function is used to encrypt data with a data element.

The external name is PDIDATE.

pty.ins_enc_date(communicationid INTEGER, dataelement VARCHAR, input_data DATE, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	DATE	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (DATE1 VARCHAR(34) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (DATE1)
   VALUES (PTY.ins_enc_date(8,'data-element',DATE('12/31/2014'),0));
```

7.4.9.2 pty.upd_enc_date

This function can be used to encrypt data with a data element.

The external name is PDUDATE.

pty.upd_enc_date(communicationid INTEGER, dataelement VARCHAR, input_data DATE, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	DATE	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (DATE1) =
   PTY.upd_enc_date(8,'data-element',DATE('07/02/1985'),0);
```

7.4.9.3 pty.sel_dec_date

This function is used to decrypt data with a data element.

The external name is PDSDATE.

pty.sel_dec_date(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(34) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as DATE.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_date(8,'data-element',DATE1,0) as DATE1
from table-name;
```

7.4.9.4 pty.ins_date

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDIDATEM.

pty.ins_date(communicationid INTEGER, dataelement VARCHAR, input_data DATE, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.

Name	Type	Description
input_data	DATE	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as DATE.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (DATE1 DATE)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (DATE1)
   VALUES (PTY.ins_date(8,'date-token',DATE('12/31/2014'),0));
```

7.4.9.5 pty.upd_date

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDUDATEM.

pty.upd_date(communicationid INTEGER, dataelement VARCHAR, input_data DATE, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	DATE	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as DATE.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (DATE1) =
   PTY.upd_date(8,'date-token’,DATE(‘07/02/1985’),0);
```

7.4.9.6 pty.sel_date

This function is used for no encryption with a data element as well as for de-tokenization.

The external name is PDSDATEM.

pty.sel_date(communicationid INTEGER, dataelement VARCHAR, input_data DATE, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	DATE	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as DATE.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_date(8,'date-token',DATE1,0) as DATE1
from table-name;
```

7.4.10 TIMESTAMP UDFs

These UDFs can be used to encrypt and decrypt TIMESTAMP data.

7.4.10.1 pty.ins_enc_timestamp

This function is used to encrypt data with a data element.

The external name is PDITSTMP.

pty.ins_enc_timestamp(communicationid INTEGER, dataelement VARCHAR, input_data TIMESTAMP, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	TIMESTAMP	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(50) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (DATETIME VARCHAR(50) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (DATETIME)
    VALUES (PTY.ins_enc_timestamp(8,'data-element',TIMESTAMP('2014-12-31 11:59:59'),0));
```

7.4.10.2 pty.upd_enc_timestamp

This function can be used to encrypt data with a data element.

The external name is PDUTSTMP.

pty.upd_enc_timestamp(communicationid INTEGER, dataelement VARCHAR, input_data TIMESTAMP, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	TIMESTAMP	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR (50) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (DATETIME) =
   PTY.upd_enc_timestamp(8,'data-element', TIMESTAMP('2014-12-31 11:59:59'),0);
```

7.4.10.3 pty.sel_dec_timestamp

This function is used to decrypt data with a data element.

The external name is PDSTSTMP.

pty.sel_dec_timestamp(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(50) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as TIMESTAMP.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_timestamp(8,'data-element',DATETIME,0) as DATETIME
```

7.4.11 TIME UDFs

These UDFs can be used to encrypt and decrypt TIME data.

7.4.11.1 pty.ins_enc_time

This function is used to encrypt data with a data element.

The external name is PDITIME.

pty.ins_enc_time(communicationid INTEGER, dataelement VARCHAR, input_data TIME, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	TIME	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (TIME1 VARCHAR(34) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (TIME1)
   VALUES (PTY.ins_enc_time(8,'data-element',TIME('11:59:59'),0));
```

7.4.11.2 pty.upd_enc_time

This function can be used to encrypt data with a data element.

The external name is PDUTIME.

pty.upd_enc_time(communicationid INTEGER, dataelement VARCHAR, input_data TIME, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	TIME	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (TIME1) =
   PTY.upd_enc_time(8,'data-element',TIME('02:59:59'),0);
```

7.4.11.3 pty.sel_dec_time

This function is used to decrypt data with a data element.

The external name is PDSTIME.

pty.sel_dec_time(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.

Name	Type	Description
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(34) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as TIME.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_time(8,'data-element',TIME1,0) as TIME1
from table-name;
```

7.4.11.4 pty.ins_time

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDITIME2.

pty.ins_time(communicationid INTEGER, dataelement VARCHAR, input_data TIME, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	TIME	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as TIME.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (TIME1 TIME)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (TIME1)
   VALUES (PTY.ins_time(8,'NOENC',TIME('11:59:59'),0));
```

7.4.11.5 pty.upd_time

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDUTIME2.

pty.upd_time(communicationid INTEGER, dataelement VARCHAR, input_data TIME, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	TIME	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as TIME.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (TIME1) =
   PTY.upd_time(8,'token-element',TIME('11:59:59'),0);
```

7.4.11.6 pty.sel_time

This function is used for no encryption with a data element as well as for de-tokenization.

The external name is PDSTIME2.

pty.sel_time(communicationid INTEGER, dataelement VARCHAR, input_data TIME, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	TIME	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as TIME.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_time(8,’token-element',TIME1,0) as TIME1
    from table-name;
```

7.4.12 INTEGER UDFs

These UDFs can be used to encrypt and decrypt INTEGER data.

7.4.12.1 pty.ins_enc_integer

This function is used to encrypt data with a data element.

The external name is PDIINT.

pty.ins_enc_integer(communicationid INTEGER, dataelement VARCHAR, input_data INTEGER, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	INTEGER	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (INT4 VARCHAR(34) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (INT4)
   VALUES (PTY.ins_enc_integer(8,'data-element',1234567890,0));
```

7.4.12.2 pty.upd_enc_integer

This function can be used to encrypt data with a data element.

The external name is PDUINT.

pty.upd_enc_integer(communicationid INTEGER, dataelement VARCHAR, input_data INTEGER, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	INTEGER	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR (34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (INT4) =
   PTY.upd_enc_integer(8,'data-element',9876543210,0);
```

7.4.12.3 pty.sel_dec_integer

This function is used to decrypt data with a data element.

The external name is PDSINT.

pty.sel_dec_integer(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(34) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as INTEGER.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_integer(8,'data-element',INT4,0) as INT4
from table-name;
```

7.4.12.4 pty.ins_integer

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDIINT2.

pty.ins_integer(communicationid INTEGER, dataelement VARCHAR, input_data INTEGER, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	INTEGER	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as INTEGER.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (INT4 INTEGER)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (INT4)
   VALUES (PTY.ins_integer(8,’integer-token(4)',1234567890,0));
```

7.4.12.5 pty.upd_integer

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDUINT2.

pty.upd_integer(communicationid INTEGER, dataelement VARCHAR, input_data INTEGER, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.

Name	Type	Description
input_data	INTEGER	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as INTEGER.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (INT4) =
   PTY.upd_integer(8,'integer-token(4)',9876543210,0);
```

7.4.12.6 pty.sel_integer

This function is used for no encryption with a data element as well as for de-tokenization.

The external name is PDSINT2.

pty.sel_time(communicationid INTEGER, dataelement VARCHAR, input_data INTEGER, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	INTEGER	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as INTEGER.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_integer(8,'integer-token(4)',INT4,0) as INT4
from table-name;
```

7.4.13 SMALLINT UDFs

These UDFs can be used to encrypt and decrypt SMALLINT data.

7.4.13.1 pty.ins_enc_smallint

This function is used to encrypt data with a data element.

The external name is PDISINT.

pty.ins_enc_smallint(communicationid INTEGER, dataelement VARCHAR, input_data SMALLINT, scid INTEGER)

Parameters

Name	Type	Description
comunicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	SMALLINT	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (INT2 VARCHAR(34) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (INT2)
   VALUES (PTY.ins_enc_smallint(8,'data-element',12345,0));
```

7.4.13.2 pty.upd_enc_smallint

This function can be used to encrypt data with a data element.

The external name is PDUSINT.

pty.upd_enc_smallint(communicationid INTEGER, dataelement VARCHAR, input_data SMALLINT, scid INTEGER)

Parameters

Name	Type	Description
comunicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	SMALLINT	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (INT2) =
   PTY.upd_enc_smallint(8,'data-element',9876,0);
```

7.4.13.3 pty.sel_dec_smallint

This function is used to decrypt data with a data element.

The external name is PDSSINT.

pty.sel_dec_smallint(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(34) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as SMALLINT.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_smallint(8,'data-element',INT2,0) as INT2
from table-name;
```

7.4.13.4 pty.ins_smallint

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDISINT2.

pty.ins_smallint(communicationid INTEGER, dataelement VARCHAR, input_data SMALLINT, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	SMALLINT	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as SMALLINT.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (INT2 SMALLINT)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (INT2)
   VALUES (PTY.ins_smallint(8,'smallint-token(2)',12345,0));
```

7.4.13.5 pty.upd_smallint

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDUSINT2.

pty.upd_smallint(communicationid INTEGER, dataelement VARCHAR, input_data SMALLINT, scid INTEGER)

Parameters

Name	Type	Description
comunicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	SMALLINT	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as SMALLINT.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (INT2) =
   PTY.upd_smallint(8,'smallint-token(2)',987,0);
```

7.4.13.6 pty.sel_smallint

This function is used for no encryption with a data element as well as for de-tokenization.

The external name is PDSSINT2.

pty.sel_smallint(communicationid INTEGER, dataelement VARCHAR, input_data SMALLINT, scid INTEGER)

Parameters

Name	Type	Description
comunicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	SMALLINT	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as SMALLINT.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_smallint(8,'smallint-token(2)',INT2,0) as INT2
from table-name;
```

7.4.14 REAL UDFs

These UDFs can be used to encrypt and decrypt REAL data.

7.4.14.1 pty.ins_enc_real

This function is used to encrypt data with a data element.

The external name is PDIREAL.

pty.ins_enc_real(communicationid INTEGER, dataelement VARCHAR, input_data REAL, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.

Name	Type	Description
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	REAL	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (REAL1 VARCHAR(34) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (REAL1)
   VALUES (PTY.ins_enc_real(8,'data-element', REAL(100.001),0));
```

7.4.14.2 pty.upd_enc_real

This function can be used to encrypt data with a data element.

The external name is PDUREAL.

pty.upd_enc_real(communicationid INTEGER, dataelement VARCHAR, input_data REAL, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	REAL	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (REAL1) =
   PTY.upd_enc_real(8,'data-element', REAL(298.981),0);
```

7.4.14.3 pty.sel_dec_real

This function is used to decrypt data with a data element.

The external name is PDSREAL.

pty.sel_dec_real(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(34) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as REAL.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_real(8,'data-element',REAL1,0) as REAL1
from table-name;
```

7.4.14.4 pty.ins_real

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDIREAL2.

pty.ins_real(communicationid INTEGER, dataelement VARCHAR, input_data REAL, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	REAL	Specifies the input data for UDF.

Name	Type	Description
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as REAL.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (REAL1 REAL)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (REAL1)
   VALUES (PTY.ins_real(8,'NOENC',REAL(100.001),0));
```

7.4.14.5 pty.upd_real

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDUREAL2.

pty.upd_real(communicationid INTEGER, dataelement VARCHAR, input_data REAL, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	REAL	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as REAL.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (REAL1) =
   PTY.upd_real(8,'NOENC',REAL(298.981),0);
```

7.4.14.6 pty.sel_real

This function is used for no encryption with a data element as well as for de-tokenization.

The external name is PDSREAL2.

pty.sel_real(communicationid INTEGER, dataelement VARCHAR, input_data REAL, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	REAL	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as REAL.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_real(8,'NOENC',REAL1,0) as REAL1
from table-name;
```

7.4.15 DOUBLE UDFs

These UDFs can be used to encrypt and decrypt DOUBLE data.

7.4.15.1 pty.ins_enc_double

This function is used to encrypt data with a data element.

The external name is PDIDBLE.

pty.ins_enc_double(communicationid INTEGER, dataelement VARCHAR, input_data DOUBLE, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	DOUBLE	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (DOUBLE1 VARCHAR(34) FOR BIT DATA)
CCSID EBCDIC IN DATABASE DSNDB04;
INSERT into table-name (DOUBLE1)
   VALUES (PTY.ins_enc_double(8,'data-element', DOUBLE(100.001),0));
```

7.4.15.2 pty.upd_enc_double

This function can be used to encrypt data with a data element.

The external name is PDUDBLE.

pty.upd_enc_double(communicationid INTEGER, dataelement VARCHAR, input_data DOUBLE, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	DOUBLE	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as VARCHAR(34) FOR BIT DATA.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (DOUBLE1) =
   PTY.upd_enc_double(8,'data-element', DOUBLE(298.981),0);
```

7.4.15.3 pty.sel_dec_double

This function is used to decrypt data with a data element.

The external name is PDSDBLE.

pty.sel_dec_double(communicationid INTEGER, dataelement VARCHAR, input_data VARCHAR FOR BIT DATA, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	VARCHAR(34) FOR BIT DATA	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as DOUBLE.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_dec_double(8,'data-element',DOUBLE1,0) as DOUBLE1
from table-name;
```

7.4.15.4 pty.ins_double

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDIDBLE2.

pty.ins_double(communicationid INTEGER, dataelement VARCHAR, input_data DOUBLE, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	DOUBLE	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as DOUBLE.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
CREATE TABLE table-name (DOUBLE1 DOUBLE)
CCSID EBCDIC IN DATABASE DSNDB04;
```

```
INSERT into table-name (DOUBLE1)
   VALUES (PTY.ins_double(8,'NOENC',DOUBLE(100.001),0));
```

7.4.15.5 pty.upd_double

This function is used for no encryption with a data element as well as for tokenization.

The external name is PDUDBLE2.

pty.upd_double(communicationid INTEGER, dataelement VARCHAR, input_data DOUBLE, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies location where policy can be found. Must be the same as configured in `pepserver.cfg`
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	DOUBLE	Specifies the input data for UDF.
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the encrypted data as DOUBLE.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
UPDATE table-name SET (DOUBLE1) =
   PTY.upd_double(8,'NOENC',DOUBLE(298.981),0);
```

7.4.15.6 pty.sel_double

This function is used for no encryption with a data element as well as for de-tokenization.

The external name is PDSDBLE2.

pty.sel_double(communicationid INTEGER, dataelement VARCHAR, input_data DOUBLE, scid INTEGER)

Parameters

Name	Type	Description
communicationid	INTEGER	Specifies the location where policy can be found. Must be the same as configured in `pepserver.cfg`.
dataelement	VARCHAR(64)	Specifies the name of data element.
input_data	DOUBLE	Specifies the input data for UDF.

Name	Type	Description
scid	INTEGER	Specifies the security co-ordinate ID, which is not used. The value should be set to zero.

Returns

This UDF returns the decrypted data as DOUBLE.

> **Note:** If the function call fails, this UDF throws an exception with an appropriate message.

Example

```
SELECT pty.sel_double(8,'NOENC',DOUBLE1,0)
    as DOUBLE1 from table-name;
```

Chapter 8

The IMS Solutions on Mainframe z/OS

The IMS Protector for z/OS is a set of products that provides capabilities for encrypting and decrypting data on IMS database. The entire database segment will be encrypted and decrypted using IMS Protector. The protector has Compression facility to compress data on segment before encryption and decompress the data after decryption.

The IMS Protector for z/OS use standard IMS segment edit/compression exit routine to invoke the encryption or decryption. These exits and their usage are documented in manuals from IBM (refer to the IBM website for downloads): *IMS Administration Guide Database Manager*. Certain restrictions from those manuals are documented in this manual, but we recommend you refer to those documents for additional information.

> **Note:** The IBM software product, z/OS Integrated Cryptographic Service Facility (ICSF), is not required by Protegrity DPS. It is not required to use any of the supported cryptographic hardware and it is not required to perform software only encryption and decryption.

Protegrity IMS Protector for z/OS consists of the following modules:

- The Cryptographic Services Manager (PTYPCSM module and its associated modules)
- The PEP server executable and its pieces which run under Open MVS (UNIX System Services or USS)
- The IMS Segment edit/compression exit routine for encrypt/decrypt functions (PTYPIMS module)
- The IMS Segment edit/compression exit routine for compress + encrypt/decrypt + decompress functions (PTYPIMSC module)

The IMS Protectors for z/OS support user access through multiple secondary authorization IDs (RACF group name). IMS Protector grant access to a user, defined in a policy created using ESA by matching the Policy role user with the system user. If match is not found, then the Policy role user will try to match with RACF group name.

> **Note:** Before you install the IMS portions of the product, you should install and customize the Protegrity for z/OS Cryptographic Servers. For more information, refer to the section *Installing the z/OS Product*.

8.1 Installing and Configuring the IMS Protector on Mainframe z/OS

The following sections explain the installation of IMS Protector Solution on Mainframe z/OS. It provides a high-level setup overview as well as post-installation verification steps.

8.1.1 Setting Up the IMS Protector

The following procedure explains how to install the IMS Protector on the Mainframe z/OS.

To setup the IMS Protector:

1. Install the PEP server. For more information about installing the PEP server, refer to the section *Installing the PEP Server*.
2. Install the Protegrity Cryptographic Services. For more information about installing the Cryptographic Services, refer to the section *Installing the Cryptographic Server*.
3. Install the IMS Protector.
   The files `PTYxxxx.IMSAUTH`, `PTYxxxx.IMSLOAD`, and `PTYxxxx.IMSSAMP` are restored using `LODxxxI.jcl`. For more information, refer to the section *Prerequisites for Installing the z/OS Product*.

`IMSAUTH`	Load module library, needs to be PDS
`IMSLOAD`	Load module library
`IMSSAMP`	RECFM F LRECL 80 JCL type library, needs to be PDS

   As per the attributes for the IMS exit routine, `IMSAUTH` must be APF authorized, or the load modules included must be copied to authorized libraries.

4. Activate the Protegrity Data Security Platform. For more information about activating the Protegrity Data Security Platform, refer to the section *Activating Protegrity Data Security Platform*.
5. Deploy a test policy. For more information about deploying a test policy, refer to the section *Deploying a Test Policy*.
6. Verify the PEP server installation. For more information about verifying the installation of the PEP server, refer to the section *Verifying the Installation*.
7. Verify the protector installation. For more information about verifying the installation of the IMS Protector on z/OS, refer to the section *Verifying the IMS Protector Installation*.

8.1.2 Verifying the IMS Protector Installation

This section describes the procedure to verify the IMS Protector installation.

Before you begin
Start the PTYCSRV procedure to start the Protegrity Cryptographic Services Manager Task. This starts the other servers. Use the procedure customized during installation. All services can be stopped by stopping the PTYCSRV procedure from an operator console.

To verify installation:

1. Submit the job `PTYxxxx.IMSSAMP`(PTYITEST) to generate the sample test Segment edit/compression exit routine, PTYIC001.
2. Re-generate the DBD with the Segment edit/compression exit routine specified in COMPRTN parameter.
3. Generate PSB (one for each LOAD and SELECT) for DBD created in step 2.
4. Run the job `PTYxxxx.IMSSAMP`(PTYPIM1) to load data into IMS segments.
5. Use File Manager to see the encrypted data.
6. Run the job `PTYxxxx.IMSSAMP` (PTYPIM2) to select and display data in output file.

8.2 Segment Edit/Compression Exit Routine

The Segment Edit/Compression exit routine allows you to encode/decode, edit, or compress the data portion of a segment.

The following database types support the Segment Edit/Compression exit routine:

- HISAM
- HDAM
- PHDAM
- HIDAM
- PHIDAM

Two types of segment manipulation are possible using the Segment Edit/Compression exit routine:

- **Data compression —** movement or compression of data within a segment in a manner that does not alter the content or position of the key field. Typically, this involves compression of data from the end of the key field to the end of the segment. When a fixed-length segment is compressed, a 2-byte field must be added to the beginning of the data portion of the segment by the user data compression routine. This field is used by IMS to determine secondary storage requirements and is the only time that the location of the key field can be altered. The segment size field of a variable-length segment cannot be compressed but must be updated to reflect the length of the compressed segment.
- **Key compression —** movement or compression of any data within a segment in a manner that can change the relative position, value, or length of the key field and any other fields except the size field. The segment size field of a variable-length segment must be updated by the compression routine to reflect the length of the compressed segment.

> **Note:** You should specify the use of the Segment Edit/Compression exit routine when you define a segment in the DBD.

8.2.1 Specifying the Segment Edit/Compression Exit Routine

This section defines the use of Segment Edit/Compression exit routine for IMS Protector.

To specify the use of the Segment Edit/Compression exit routine, use the “COMPRTN= keyword” of the SEGM statement in the DBD.

```
DBD NAME=LIBDBDC,ACCESS=(HDAM),RMNAME=(DFSHDC40,1,5,200)
DATASET DD1=INDD
SEGM NAME=LIBSEG,PARENT=0,BYTES=30,COMPRTN=(PTYIC001,KEY)
FIELD NAME=(LIBID,SEQ,U),BYTES=5,START=1,TYPE=C
FIELD NAME=ISSDATE,BYTES=10,START=6,TYPE=C
FIELD NAME=RETDATE,BYTES=10,START=16,TYPE=C
FIELD NAME=STUID,BYTES=5,START=26,TYPE=C
DBDGEN
FINISH
END
```

8.2.2 IMS Protector Restrictions and Limitations

This section lists the restrictions and limitations of the IMS Protector.

1. The DFSRRC00 routine is compiled in authorized mode and provided with the product, which needs to reside in authorized library. The DBD and PSB libraries should also should be authorized libraries.
2. Only the CUSP encryption logics are supported.
3. The Segment Edit/Compression exit routine cannot be defined in a logical database DBD.
4. Any segment type can be edited or compressed (using either data or key compression) if the segment is:
   - Not a logical child
   - Not in an HSAM, SHISAM, or index database
5. Data compression is allowed but key compression is not allowed when the segment is:
   - A root segment in a HISAM database
   - A segment in a DEDB database

6. For Variable segments, using segment edit/compression exit routine, Maximum length should be increased by 1 byte to hold the flag value.
7. On error, the routine will return to IMS with Abend code 2990 (ABENDU2990) and corresponding return code and reason code in Register 15. Unhandled processing error will result in abend 840.
8. When using the IMS Segment Edit/Compression exit routine with the HISAM database, the Overflow file record size should be increased to accommodate the intermediate addition of the child segments.

8.2.3 Usage of Compression and Decompression Facility

The IMS segment edit/compression exit routine allows compression and decompression of customer data.

The customer site can choose to compress the data, in addition to encryption, while using Protegrity's segment edit/compression exit routine. During LOAD, INSERT, and UPDATE, the data will be compressed first, using the MVS standard compression macro CSRCESRV, and then the compressed data would be encrypted. During UNLOAD and SELECT, the data will be decrypted first, and then the decrypted data will be decompressed using the MVS standard compression macro CSRCESRV.

The CSRCESRV exploits the occurrence of repeated characters in a data stream. The encoded data contains a combination of symbols that represent strings of repeated characters and the original characters that are not repeated.

For more details on MVS standard compression macro CSRCESRV operations, refer to the *z/OS MVS Programming: Assembler Services Guide*.

To compress the data using Protegrity’s segment edit/compression exit routine, the site must configure the parameter "COMPRESS=Y" in the customized segment edit/compression exit routine that is generated with the PTYITEST sample job. The default value is "COMPRESS=N".

> **Note:** With non-compressible data as input, Protegrity’s segment edit/compression exit routine, will not result in error, rather encrypt the original input data as is.

> **Note:**
>
> Samples to explain how data protection and unprotection can be performed during LOAD/INSERT/UPDATE and UNLOAD/SELECT respectively, using IMS Segment Exit/Compression Routines are mentioned in the section *Appendix H- Sample Code for the IMS Protector Mainframe*.

8.2.4 Return and Reason Codes

The IMS segment edit/compression exit routine becomes a part of the IMS control or batch region, any abnormal termination of this routine terminates the entire IMS region. Any user-written segment edit/compression exit routine should return to IMS with an abend code and a reason code instead of initiating a standard abend.

Protegrity IMS Protector should return to IMS with Abend Code 2990. The return and reason codes are provided in register 15.

The following convention is used:

- the first 2 bytes of register 15 should have “PY” to identify that the abend is generated by Protegrity IMS protector
- the 3rd byte contains the return code
- the 4th byte contains the reason code

The IMS segment edit/compression exit routine return codes and reason codes have a 4 bytes binary format. The following table describes these codes.

Table 8-1: Return and Reason Codes

Error	Return Code Dec.	Reason Code Dec.	Return and Reason Code
Access denied	4	6	X’D7E80406’
Access denied - Unknown user	4	1	X’D7E80401’
Access denied - Unknown data element	4	2	X’D7E80402’
Access denied - User has no privileges for data element	4	3	X’D7E80403’
Access denied - User has no privileges for data element for the current time period	4	4	X’D7E80404’
Function address null	12	10	X’D7E80C0A’
Clear text length address null	12	11	X’D7E80C0B’
Clear text address null	12	12	X’D7E80C0C’
Cipher text length address null	12	13	X’D7E80C0D’
Cipher text address null	12	14	X’D7E80C0E’
Data element address null	12	15	X’D7E80C0F’
Data Protector not active	12	17	X’D7E80C11’
Data Protector not active	12	23	X’D7E80C17’
Product not active	12	30	X’D7E80C1E’
Invalid function code	12	16	X’D7E80C10’
Policy not ready	12	31	X’D7E80C1F’
Clear text length zero	12	41	X’D7E80C29’
Cipher text length zero	12	42	X’D7E80C2A’
Cipher text length less than clear text length	12	43	X’D7E80C2B’
Peps policy locked	12	44	X’D7E80C2C’

Error	Return Code Dec.	Reason Code Dec.	Return and Reason Code
Query null data element	12	45	X’D7E80C2D’
Invalid checksum	12	47	X’D7E80C2F’
Invalid padding	12	48	X’D7E80C30’
Invalid IV	12	49	X’D7E80C31’
Cipher text area too small	12	50	X’D7E80C32’
Plain text area too small	12	51	X’D7E80C33’
Checksum validation failure	12	52	X’D7E80C34’
Unsupported algorithm	12	88	X’D7E80C58’
Unknown algorithm	12	93	X’D7E80C5D’
Default data element not supported	12	94	X’D7E80C5E’
Abend percolated	12	251	X’D7E80CFB’
Input length for compression less than 2	12	61	X’D7E80C43’
Input length for expansion less than 2	12	62	X’D7E80C44’
Expansion service error	12	63	X’D7E80C45’
Sequence Field not in Segment	12	65	X’D7E80C47’
Segment length is negative	12	66	X’D7E80C48’

8.3 *PTYPSLL/PTYPSLI* to protect/unprotect IMS data

In addition to DBDEXITs, z/OS Stateless Application Protector, PTYPSLL and PTYPSLI, can be used to protect/unprotect the data in IMS database.

You should have the libraries of z/OS Application Protector. For more information about Application Protector on z/OS, refer to the section *Application Protector on Mainframe z/OS*.

> **Note:**
> Samples to explain how IMS data can be protected and unprotected using zOS Application protector, *PTYPSLL/PTYPSLI*, are mentioned in the section *Appendix E- z/OS Application Protector Samples*.

Appendix A

Uploading Certificates from ESA to z/OS Manually

This section describes how to upload certificates from the ESA to z/OS manually.

To upload certificates from the ESA to z/OS manually:

1. Download the *.tar* file of certificates from the following link to your local machine, using the ESA admin credentials.
   https://<ESA External IP>:8443/dps/v1/deployment/certificates
2. Run the following FTP commands to manually upload the *.tar* files to z/OS:

```
cd <path where tar file is downloaded>  <- path on your PC
ftp 1.1.1.1                                  <- z/OS IP address or host name
…                                           <- provide TSO userid and password as
prompted
cd <install dir>/defiance_dps/data           <- install directory path
bin                                          <- transfer files in binary mode
put *.tar                                    tar file of certificates
quit
```

3. In z/OS Open MVS, the *.tar* files of the certificates are available in the *<install dir>/defiance_dps/data* directory. Run the following commands to untar the certificate files:

```
tar xvf *.tar                                    <- untar the certificates
                                                       <- Files after untar:
                                                       CA.pem, cert.jks, cert.key,
cert.pem, certkeyup.bin
iconv -f ISO8859-1 -t IBM-1047 CA.pem > CA.pem.conv     <- Converting PEM certificates
iconv -f ISO8859-1 -t IBM-1047 cert.pem > cert.pem.conv
iconv -f ISO8859-1 -t IBM-1047 cert.key > cert.key.conv
mv CA.pem.conv  CA.pem                                   <- moving converted PEM
certificates to ../data folder
mv cert.pem.conv  cert.pem
mv cert.key.conv  cert.key
```

> **Note:** Codepage mentioned in PEM certificates conversion commands, can be specified as per client requirement.

Appendix B

Sample Codes for Automated Installation

This section contains the following sample codes for automated installation.

10.1 Sample JCL to Execute the Script to Update the *pepserver.cfg* file

The following sample JCL can be used to update the parameters in the `pepserver.cfg` file.

```
//PTYRCFG  JOB ,MSGCLASS=H,NOTIFY=&SYSUID
//JCLREXX  EXEC PGM=IKJEFT01
//*  LOADLIB FOR REXX
//SYSEXEC  DD DSN=PTYBLD.STAGE701.CNTL,DISP=SHR
//* ISPF LIBRARIES
//ISPPROF  DD DSN=&&ISPPROF,DISP=(,PASS),
//             UNIT=SYSDA,SPACE=(CYL,(1,1,10),RLSE),
//             RECFM=FB,LRECL=80
//ISPPLIB  DD DISP=SHR,DSN=USER.ISPPLIB
//ISPMLIB  DD DISP=SHR,DSN=ISP.SISPMENU
//ISPSLIB  DD DISP=SHR,DSN=ISP.SISPSLIB
//ISPTLIB  DD DISP=SHR,DSN=ISP.SISPTENU
//ISPLOG   DD DUMMY
//*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN  DD *
 ISPSTART CMD(PTYXCFG <Input file> <Loadlib> )
```

The ***ISPSTART*** command requires the following three parameters to be passed.

PTYXCFG	REXX script to update the parameters in `pepserver.cfg` file. The script is provided in `PTYxxxx.SPTYSAMP` files.
Input file	Input file which mentions the changes to be done in `pepserver.cfg` file. Sample file (`PTYICFG`) is provided in `PTYxxxx.SPTYSAMP`. For more details, refer the following note.

Loadlib	Loadlib contains the executable for assembler program (*PTYPCFG*). The loadlib is provided in the *PTYxxxx.SPTYAUTH* library.

Note: Details of the Sample Input file.

```
dsn = /opt/protegrity/dps6522/defiance_dps/data/pepserver.cfg
level = ALL
communicationid = 1
```

dsn	The complete path for the *pepserver.cfg* file should be updated. This is mandatory.
parameters	Mention the parameters to be updated in *pepserver.cfg* file, in different rows.

10.2 Sample Code to use the *BPXBATCH* to Install the PEP Server

The following sample JCL can be used to install the PEP Server.

```
//PEPINST  JOB ,MSGCLASS=H,NOTIFY=&SYSUID
//*
//*--------------------------------------------------------------------
//* PEPINST  - INSTALL THE PROTEGRITY POLICY ENFORCEMENT POINT SERVER
//*
//*--------------------------------------------------------------------
//*
// SET HLQ=%ptyhlq       --> FROM BUILD
// SET INSTLDIR='/usr/local/pty'
// SET INPPATH='/usr/local/pty/dps700/pepserver.install.input'
//CHKDISK  EXEC PGM=IKJEFT01,
//            PARM='%RXDSKSPC &INPPATH'
//SYSEXEC  DD DSN=&HLQ..SPTYSAMP,DISP=SHR
//*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN  DD *
//*
//PEPINST  EXEC PGM=BPXBATCH,REGION=0M,TIME=NOLIMIT,COND=(4,LE),
//      PARM='sh'
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//STDENV   DD  DISP=SHR,DSN=&HLQ..SPTYSAMP(PTY$ENV)
//STDIN    DD  PATHDISP=KEEP,PATHOPTS=(ORDONLY),
//             PATH='&INSTLDIR/dps700/pepserver.install.input'
//*
```

10.2 Sample Code of *pepserver.install.input*

```
/usr/local/pty/dps700/PepServerSetup_zOS_Mainframe_7.0.1.n.sh
y
/opt/protegrity/dps700
10.10.3.104
```

10.3 Details on the JMVS jcl to Install the Cryptographic Server

This section provides details about the *JMVS* jcl on how to install the Cryptographic Server.

The following are the members added to *PTYxxxx.SPTYSAMP*:

JMVS	JCL has two steps –

	JCREATE updates parameters in started tasks and parameter file; JCOPY copies them to procedure library (*USER.PROCLIB*) and *auth.SPTYSAMP* JCREATE uses RXCREATE and JCOPY uses IDCAMS utility.
RXCREATE	REXX script to update parameters in started tasks and PTYPARM file
SKELSPEP	Skeleton jcl used in RXCREATE for updating parameters in PTYSPEP task
SKELPPEP	Skeleton jcl used in RXCREATE for updating parameters in PTYPPEP task
SKELCSRV	Skeleton jcl used in RXCREATE for updating parameters in PTYCSRV task
SKELPARM	Skeleton jcl used in RXCREATE for updating parameters in PTYPARM file

Note: Nomenclature used - Members starting with RX are the REXX scripts, J are the JCLs and SKEL are the skeleton JCLs used in REXX scripts.

In the JMVS, SET statements need to be customized. Also, parameters need to be mentioned in ISPSTART in SYSTSIN step of JCREATE.

The JCREATE will create the jobs and put them in *PTYxxxx.SPTYSAMP* with names as *PTYSPEPS*, *PTYPPEPS*, *PTYCSRV*, and *PTYPARM*.

The JCOPY will copy the jobs created in *PTYxxxx.SPTYSAMP* to procedure library, *USER.PROCLIB*.

Appendix C

Sample Code for the *PTYPARM* Configuration file

This section contains sample code for the *PTYPARM* configuration file.

The contents of the *ptyparm* configuration file that is used for customizing the Cryptographic Servers on the Mainframe z/OS platform, is listed below.

```
Protegrity Defiance Data Protection System (DPS)
Data Protector Startup Parameters

  Syntax rules: 1) Blank lines are ignored.
                2) Comment lines indicated by * or a blank in column 1.
                3) Text starting in column 1 on each non-comment line
                   constitutes the parameter specification and is
                   delimited by the first blank character.  Any text
                   after the first blank is considered a comment.
                4) Multiple parameters may be specified on the same
                   line by separating them with commas with no
                   intervening blanks.

START=AUTO          Startup type: AUTO | WARM | COLD

COMMID=44           Communication id used by the PEP Server This value
                    must match the value specified by the communicationid
                    parameter in the PEP Server configuration file,
                    pepserver.cfg.

NOCST               New default is no CST
* CSTPROC=PTYCST      Cryptographic Service Task (CST) startup procedure

PEPSPROC=PTYSPEPS   PEP Server (PEPS) startup procedure

PEPPPROC=PTYPPEPS   PEP Server shutdown procedure

LOGQBS=10           Log buffer queue block size in megabytes
LOGMQB=123          Maximum allowed log buffer queue blocks
LOGRCDI=10          LOG quiesce record count display interval in seconds
*
* Next three values deal with moving log records from MVS to PepServer.
*   LOGSHMPCT       is the maximum percent MVS protectors should fill
*                   logging shared memory.  Can be 100% if UDFs and XC
*                   are not used.
*   LOGMAXLOCKTIME  While PTYCSRV is moving records from the MVS side
*                   protectors into shared memory, PepServer cannot send
*                   log records to ESA and UDFs/XC cannot add log records
*                   Generally it is safeest to leave this value alone
*   LOGMAXDELAY     A sub-task of PTYCSRV moves the records into shared
*                   memory for PepServer.  This is the maximum timer
*                   delay for that sub-task to wait before trying to
```

```
*                  move more log records into shared memory.
*
LOGSHMPCT=15       Maximum to fill log shared memory at one pass
LOGMAXLOCKTIME=50  Time in milliseconds to lock log shared memory
LOGMAXDELAY=200    Maximum delay before moving log records in ms

PEPSTIME=240       Time to wait for PEP Server startup in seconds



USERNAME=(USERID,GROUPNAME)  Use RACF group name if userid <> DPS user
*USERNAME=USERID             Use RACF userid only; do not use group name
*USERNAME=GROUPNAME          Use RACF group name only; do not use userid

FRCODEPG=01047               Unicode Conversion service - From Codepage
TOCODEPG=00923               Unicode Conversion service - To Codepage

DB2AUTHIDLVL=1               Number of DB2 Secondary AUTHIDs to check

PEPLOGPATH=<install dir>/defiance_dps/data/pepserver.log   Path for pepserver.log
FILE_CONTFAILREC=NO          Do not continue in case of an error
```

The *LOGQBS* is the size of the buffer that we do a *STORAGE* macro for. The *LOGMQB* is the number of those buffers that we will get in total. So, in the sample case we will get 210MB buffers at startup and we will get up to an additional 121 buffers for a total of 1,230 MBs of log buffers. The values can only be changed at *PTYCSRV* startup and cannot be dynamically changed. However, “f ptycsrv,d log” will show how many buffers are in use.

Default use is that the Protegrity data policy will be applied using the External Security Manager *USERID* first. If the requesting TCB user is not in the policy, then the primary *GROUPNAME* assigned by the ESM will be tried. If neither is in the policy, then the attempt to do ciphering will be rejected. The site can modify this default by changing the *USERNAME* parameter to only user *USERID* or only use *GROUPNAME* or to user *GROUPNAME* first and *USERID* if the *GROUPNAME* is not in the policy.

The *PTYCSRV* started task needs to wait for the PEP Server to make the policy available. Depending on several factors including the Open MVS priority and the size of the policy, this can take several seconds. The *PEPSTIME* allows the site to change how long the *PTYCSRV* will wait before cancelling itself because the PEP Server has not made the policy available.

Unicode conversion services support has been provided for Token data elements. The parameter *FRCODEPG* (From Codepage) specifies the System codepage of the site and should be changed by the System administrator while setting up PTYPARM file. The parameter *TOCODEPG* (To Codepage) is to be used by token conversion routines, the default value supported is Codepage: CCSID 923 Name - ISO 8859-15 ASCII.

> **Note:** We do not suggest sites to change the *TOCODEPG* parameter as this can result in discrepancy between the tokenized values returned by Database protectors Fieldproc, Editproc and UDF's.

The *DB2AUTHIDLVL* parameter is used to identify the number of secondary authorization IDs that must be matched with the DB2 Database protectors. The default value for this parameter is 1, which will invoke default behavior of matching Policy role user with DB2 User ID. If the value is not matched, then the Policy role user will be matched with only one DB2 secondary authorization ID. For example, if the site changes the value of this parameter to 5, then Policy role user will be matched with DB2 User ID. If the value is not matched, then the Policy role user will be matched with first 5 occurrences of DB2 secondary authorization ID.

The *PEPLOGPATH* parameter is used to provide the path for `pepserver.log` file. Using this parameter, messages from the `pepserver.log` file with logging level= WARNING/ERROR/SEVERE are displayed in system logs during PTYCSRV start-up. This is an optional parameter.

> **Note:** It is recommended to use the *PEPLOGPATH* parameter when ***append=no*** is set in the logging configuration section of the `pepserver.cfg` file, else the **WARNING/ERROR/SEVERE** messages from the previous *PTYCSRV* executions will be captured in the system logs.

The *FILE_CONTFAILREC* parameter is added to configure the setting that displays the fail record information in the job log. The default value is *FILE_CONTFAILREC=NO*,that does not allow processing of subsequent records after a failed record, whereas, *FILE_CONTFAILREC=YES* allows the processing of subsequent records after a failed record. This is an optional parameter.

Appendix D

Sample RACF Setup for the Started tasks

This section contains sample code for RACF setup for the started tasks.

The following sample code is a setup for the RACF permissions in the following started tasks:

- Protegrity started tasks for the Cryptographic Services Manager
- PEP Server started tasks (starting and stopping the PEP Server)

> **Note:** The setup in the sample code below are suggestions only

```
AG   PPPSGRP SUPGROUP(SYS1) OWNER(SECURITY)
ALG  PPPSGRP DATA('Protegrity Protection Server Group')
ALG  PPPSGRP OMVS(GID(0000992058))

AU   PPPS DFLTGRP(PPPSGRP) OWNER(SECURITY)
ALU  PPPS NOPASSWORD NOOIDCARD
ALU  PPPS NAME('Protegrity Protection Server')
ALU  PPPS DATA('Protegrity Protection Server')
ALU  PPPS OMVS(UID(0000992058)                                          +
               HOME('/opt/protegrity')                                  +
               PROGRAM('/bin/sh'))

CO   PPPS GROUP(PPPSGRP) OWNER(SECURITY)

AD   'SYS6.PPPS.*.**' GEN UACC(NONE) OWNER(SECURITY)
ALD  'SYS6.PPPS.*.**' GEN AUDIT(FAILURES(READ))
PE   'SYS6.PPPS.*.**' ID(*)      ACCESS(READ)  GEN
PE   'SYS6.PPPS.*.**' ID(TSUP)   ACCESS(ALTER) GEN
PE   'SYS6.PPPS.*.**' ID(PPPSGRP) ACCESS(ALTER) GEN

AD   'SYS3.PPPS.*.**' GEN UACC(NONE) OWNER(SECURITY)
ALD  'SYS3.PPPS.*.**' GEN AUDIT(FAILURES(READ))
PE   'SYS3.PPPS.*.**' ID(*)      ACCESS(READ)  GEN
PE   'SYS3.PPPS.*.**' ID(PPPSGRP) ACCESS(READ) GEN
PE   'SYS3.PPPS.*.**' ID(TSUP)   ACCESS(ALTER) GEN
SETROPTS GENERIC (DATASET ) REFRESH

PE   BPX.DAEMON    CLASS(FACILITY) ID(PPPS) ACCESS(READ)
PE   BPX.SUPERUSER CLASS(FACILITY) ID(PPPS) ACCESS(READ)
SETROPTS RACLIST (FACILITY) REFRESH

RDEF STARTED PTYCSRV.* UACC(NONE) OWNER(SECURITY)
RALT STARTED PTYCSRV.* AUDIT(FAILURES(READ))
RALT STARTED PTYCSRV.* STDATA(USER(PPPS) TRUSTED(NO)                    +
     PRIVILEGED(NO) TRACE(NO))
```

```
RDEF STARTED PTYPPEPS.* UACC(NONE) OWNER(SECURITY)
RALT STARTED PTYPPEPS.* AUDIT(FAILURES(READ))
RALT STARTED PTYPPEPS.* STDATA(USER(PPPS) TRUSTED(NO)                  +
     PRIVILEGED(NO) TRACE(NO))

RDEF STARTED PTYSPEPS.* UACC(NONE) OWNER(SECURITY)
RALT STARTED PTYSPEPS.* AUDIT(FAILURES(READ))
RALT STARTED PTYSPEPS.* STDATA(USER(PPPS) TRUSTED(NO)                  +
     PRIVILEGED(NO) TRACE(NO))

SETROPTS RACLIST (STARTED ) REFRESH
```

Appendix E

z/OS Application Protector Samples

Several sample programs are available to show how to use the various interface programs of the Application Protector. This Appendix documents the interface and the associated sample programs available.

Most of the samples are written in COBOL. The samples are designed to be compiled by any level of the z/OS IBM COBOL compiler. They are not reliant upon having any specific level of COBOL. If you compile the samples of the IBM COBOL, version 5 or newer, then the load library needs to be a PDS-E, and not PDS. The PTYnnnnn.APSLOD is the PDS-E load library.

The sample programs are in the PTYnnnnn.APSAMP library and the associated JCLs are in the PTYnnnnn.APSCNTL library. The JCLs with *PTYJ* as prefix, are COMPILE JCLs, and the JCLs with *PTYR* as prefix, are RUN JCLs.

13.1 Test Data

The following are some test data files that are included in the PTYnnnnn.APSAMP library in XMIT format. Use the TSO RECEIVE command to reload the data.

1. The APDATA1 contains a few simple records with different character datatypes (alpha, numeric, and alpha-numeric) that allows you to explore the different token algorithms.
2. The CFUDATAF contains approximately one thousand records with 16-character numeric data, similar to credit card numbers, last names, and first names.
3. The APDATAIM contains a few simple records with segments defined as per the sample IMS database. All the IMS samples are available with respect to the sample library database.

13.2 Copybooks

The following copybooks are included in the PTYnnnnn.APSAMP library:

- **PTYCCSIP** – to be included in the samples that are run for PTYPSLL or PTYPSLI.

- **PTYCCXIP** – to be included in the samples that are run for PTYPCXL or PTYPCXI.
- **PTYIMINP** – to be included in the samples that are run for IMS data with PTYPSLL and PTYPCXL.
- **PTYIMPCB** – to be included in the samples that are run for IMS data with PTYPSLL and PTYPCXL.

13.3 Compilation of programs

The following compile JCLs are available:

- PTYCMPL – This will compile all the non-IMS COBOL programs.

  For more information about the IMS sample compilations, refer to the table *F-1: Sample Programs*.

- Each program(s) has its own compile JCL, as mentioned in the table *Sample Programs*.
- The PTYJIMS1 will compile all the IMS sample programs. These samples are for using the Application Protector with IMS data, and not samples for the Protegrity IMS Protector. For IMS samples, the DBDs and PSBs need to be generated and load modules to be kept in authorized PDS.
  a. The sample DBD, PTYnnnnn.APSAMP(PTYIMDBD) is generated using the sample JCL, PTYnnnnn.APSCNTL(PTYGNDBD). The compiled DBD should be available in the authorized PDS.
  b. The sample PSBs, PTYnnnnn.APSAMP(PTYIMPSB) for INSERT and PTYnnnnn.APSAMP(PTYIMPSS) for SELECT, are generated using the sample JCL, PTYnnnnn.APSCNTL(PTYGNPSB). The compiled PSBs should be available in the authorized PDS.

The following table describes the sample programs and corresponding COMPILE and RUN JCLs.

Table E-1: Sample Programs

No.	COBOL Program	COMPILE JCL	RUN JCL	Short Description
1	ENCDEC	PTYJEDC	PTYREDC	Protect and unprotect using PTYPSLL or PTYPSLI
2	ENCDECL1	PTYJEDL1	PTYREDL1	Protect and unprotect using PTYPSLL or PTYPSLI
3	ENCDECX1	PTYJEDX1	PTYREDX1	Protect and unprotect using PTYPCXL or PTYPCXI
4	PTYPAP1	PTYJAP1	PTYRAP1	Protect data (PTYPSLL or PTYPSLI) and write it to VSAM file
5	PTYPAP2			Unprotect the protected VSAM file (PTYPSLL or PTYPSLI) and write the unprotected data into a sequential file
6	PTYPAX1	PTYJAX1	PTYRAX1	Only protection using PTYPCXL or PTYPCXI
7	PTYPAX2			Only unprotection using PTYPCXL or PTYPCXI
8	PTYPSL1	PTYJSL1	PTYRSL1	Only protection using PTYPSLL or PTYPSLI
9	PTYPSL2			Only unprotection using PTYPSLL or PTYPSLI
10	PTYQCKID	PTYJQCK	PTYRQCK	Query Current keyID; PTYPCXL or PTYPCXI
11	PTYQDDES	PTYJQDS	PTYRQDS	Query Default data element; PTYPSLL or PTYPSLI
12	PTYQDDEX	PTYJQDX	PTYRQDX	Query Default data element; PTYPCXL or PTYPCXI
13	PTYQDKID	PTYJQDK	PTYRQDK	Fetch Data keyID; PTYPCXL or PTYPCXI
14	PTYVSAM	PTYJVSM	PTYRVSM	Read VSAM FILE and write protected data to another VSAM file; PTYPSLL or PTYPSLI
15	PTYVSAM1	PTYJVSM1	PTYRVSM1	Protect and unprotect VSAM files (not PRI-KEY) using length preserving algorithms; PTYPSLL or PTYPSLI
16	PTYVSAM2	PTYJVSM1	PTYRVSM2	Protect and unprotect VSAM files (not PRI-KEY) using non-length preserving algorithms; PTYPSLL or PTYPSLI
17	PTYVSAM3	PTYJVSM1	PTYRVSM3	Write protected data (including PRI-KEY) in VSAM file using length preserving algorithms; PTYPSLL ot PTYPSLI

No.	COBOL Program	COMPILE JCL	RUN JCL	Short Description
18	PTYVSAM4			Unprotect protected data (including PRI-KEY) in VSAM file using length preserving algorithms; PTYPSLL or PTYPSLI
19	PTYIMS1	PTYJIMS1	PTYRIMS1	Protect data (PTYPSLL or PTYPSLI) before inserting to IMS database
20	PTYIMS2			Unprotect data (PTYPSLL or PTYPSLI) after selecting data from IMS database
21	PTYIMS3		PTYRIMS2	Protect data (PTYPCXL or PTYPCXI) before inserting to IMS database
22	PTYIMS4			Unprotect data (PTYPCXL or PTYPCXI) after selecting data from IMS database
23	PTYSLLDB	PTYJSLDB	PTYRSLDB	Protects the data using PTYPSLL or PTYPSLI before inserting the protected data into DB2 table. Unprotects the data into a file using PTYPSLL or PTYPSLI after selecting the protected data from DB2 tables.
24	PTYPSC1	PTYJSC1	PTYRSC1	PTYPSC1 shows how to use the alternate user ID variable.
25	PTYPSEIV	PTYJEDL1	PTYRSEIV	Protect and unprotect with support of external IV parameter using PTYPSLL or PTYPSLI.
26	PTYPCEIV	PTYJEDL1	PTYRCEIV	Protect and unprotect with support of external IV parameter using PTYPCXL or PTYPCXI.

13.4 Interfaces

There are two types of interfaces. Each type includes the following two interfaces:

- A Language Environment (LE) capable interface that utilizes LE Macros or functions to get memory, etc.
- A non-LE interface that uses the operating system capabilities for memory and other purposes.

The two interfaces have the same calling sequence and parameters. They may be used interchangeably just by changing the name of the program called. The non-LE interface can be called by an LE program, but it is slightly less efficient than the LE capable version, when called by a LE program.

For more information about the interfaces, refer to the sections *Stateless Interfaces (APIs): PTYPSLI, PTYPSLL, and PTYPSLC* and *Stateful Interfaces: PTYPCXI and PTYPCXL*.

The following are the interfaces.

1. **Stateless Interfaces:** The PTYPSLL and PTYPSLI interfaces are stateless. Each call is atomic. The entire transaction is processed before returning to the caller. The protection or unprotection is performed on the data and an audit record is generated before the return. The PTYPSLL interface is Language Environment (LE) capable and it uses LE functions to get its dynamic storage. The PTYPSLI interface is LE tolerant. It may be called from LE programs. The PTYPSLI interface uses assembler MACROs to get its dynamic storage.
2. **Stateful Interfaces:** The PTYPCXL and PTYPCXI interfaces are stateful. These are called first to create a *session* that causes a control block to be allocated and a handle to be returned to the caller. A part of this session creation turns off successful audit logging until the session is ended. Then one successful audit log of protect and one of unprotect are created giving a count of the transactions. Each subsequent call then uses the handle to find the control block and do the requested operation. At the end, a disconnect call is done to free the control block and cause the audit logs to be created. The PTYPCXL interface is LE capable and the PTYPCXI interface is non-LE capable.

13.5 Detailed Descriptions of COBOL programs and JCLs

All the programs accept the data element name from the SYSIN DD. All the JCLs have various sets to assign high-level qualifiers so that multiple JCL statements do not need to be changed. The latest compiles of all the programs should be in PTYxxxxx.APSLOD for non-IMS programs and the PTYxxxxx.APSAUTX (need to be APF-authorised) for IMS programs. The COBOL compiler version 5.2.0 is used for the compiles.

13.5.1 ENCDEC

This ENCDEC COBOL program protects and unprotects 11 characters of data. It uses the CFUDATAF data file as input. Input is DDNAME INDATA. The protected output is written to DDNAME CIPHERA and unprotected output is written to DDNAME OUTDATA. The program in the same library uses PTYPSLL as its interface program. The ENCDEC accepts the data element from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the sample JCLs.

a. The PTYJEDC is the compile JCL. It uses COBOL V5R2, but the program will also compile with COBOL V4 or V6.

b. The PTYREDC is the run JCL. It creates the following output files:

   - CIPHER
   - CLEAR

   The input and output files are printed. The input and CLEAR files are compared as the last job step. The compare step should have a return code of zero showing that all records were equal.

13.5.2 ENCDECL1

This ENCDECL1 COBOL program is parameter driven. It uses the stateless interface by calling it dynamically instead of calling statically. The parameters are passed through the COBOL ACCEPT verb and DDNAME SYSIN. Data element to be used, start position, and length are part of the parameters. You need to check the beginning of the Procedure Division to see the parameters and their order. The example runtime also shows the parameters. The ENCDECL1 uses the CFUDATAF input file.

The following are the sample JCLs.

a. The PTYJEDL1 is the compile JCL. The default is COBOL V5R2, but the program will also compile with COBOL V4 or V6.

b. The PTYREDL1 is the runtime JCL. As shipped, the PTYPSLL is the interface program specified in the parameters. There is a print step for the input, cipher, and clear files. There is also a compare step for the input and clear files.

13.5.3 ENCDECX1

This ENCDECX1 COBOL program is parameter driven. It uses the stateful interface by calling it dynamically instead of calling statically. The parameters are passed via the COBOL ACCEPT verb and DDNAME SYSIN. Data element to be used, start position, and length are part of the parameters. You need to check the beginning of the Procedure Division to see the parameters and their order. The example runtime also shows the parameters. The ENCDECX1 uses the CFUDATAF input file.

The following are the sample JCLs.

a. The PTYJEDX1 is the compile JCL. The default is COBOL V5R2, but the program will also compile with COBOL V4 or V6.

b. The PTYREDX1 is the runtime JCL. As shipped, the PTYPCXL is the interface program specified in the parameters. There is a print step for the input, cipher, and clear files. There is also a compare step for the input and clear files.

13.5.4 PTYPAP1 and PTYPAP2

These PTYPAP1 and PTYPAP2 COBOL programs use the APDATA1 input file to build a VSAM file. The PTYPAP1 reads the input and creates the protected VSAM file. The PTYPAP2 reads the VSAM file and creates a clear-text output file. Both the programs use the PTYPSLI stateless interface. As written, the credit card number is protected.

The following are the sample JCLs.

a. The PTYJAP1 is the compile JCL. Both PTYPAP1 and PTYPAP2 are compiled by this JCL. Default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.

b. The PTYRAP1 is the runtime JCL. The JCL has multiple steps. It defines the VSAM file using IDCAMS, runs PTYPAP1 to load the VSAM file, and runs PTYPAP2 to create the clear file. The JCL then uses IDCAMS to print the three files (input, VSAM, and clear) and compares the input and clear files.

13.5.5 PTYPAX1 and PTYPAX2

These PTYPAX1 and PTYPAX2 COBOL programs use CFUDATAF input file. The PTYPAX1 reads the input and creates the protected file. The PTYPAX2 reads the protected file and creates a clear-text output file. Both the programs use any of the stateful interfaces. The data element is accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the sample JCLs.

a. The PTYJAX1 is the compile JCL. Both PTYPAX1 and PTYPAX2 are compiled by this JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.

b. The PTYRAX1 is the runtime JCL. The JCL has multiple steps. It creates the following output files:

   - CIPHER
   - CLEAR

   The PTYPAX1 protects the input file and creates the CIPHER file. The PTYPAX2 unprotects the CIPHER file and writes it to the CLEAR file. The input and both the output files are printed. The input and CLEAR files are compared as the last job step. All the records should be equal.

13.5.6 PTYPSL1 and PTYPSL2

These PTYPSL1 and PTYPSL2 COBOL programs use CFUDATAF input file. The PTYPSL1 reads the input and creates the protected file. The PTYPSL2 reads the protected file and creates a clear-text output file. Both the programs use any of the stateless interfaces. The data element is accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYJSL1 is the compile JCL. Both PTYPSL1 and PTYPSL2 are compiled by this JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.

b. The PTYRSL1 is the runtime JCL. The JCL has multiple steps. It creates the following output files:

   - CIPHER
   - CLEAR

   The PTYPSL1 protects the input file and creates the CIPHER file. The PTYPSL2 unprotects the CIPHER file and writes it to the CLEAR file. The input and both output files are printed. The input and the CLEAR files are compared as the last job step. All the records should be equal.

13.5.7 PTYQCKID

This PTYQCKID COBOL program queries the current KEYID and writes the fetched value in the CXI-Reason-Code. It uses any of the stateful interfaces. The API and data element are accepted from DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYJQCK is the compile JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.

b. The PTYRQCK is the runtime JCL. The API and data element are passed in SYSIN.

13.5.8 PTYQDDES

This PTYQDDES COBOL program queries the default data element and writes the fetched value in the CSI-Reason-Code. It uses any of the stateless interfaces. The API and policy name are accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYJQDS is the compile JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.

b. The PTYRQDS is the runtime JCL. The API and policy name are passed in SYSIN.

13.5.9 PTYQDDEX

This PTYQDDEX COBOL program queries the default data element and writes the fetched value in the CXI-Reason-Code. It uses any of the stateful interfaces. The API and policy name are accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYJQDX is the compile JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.

b. The PTYRQDX is the runtime JCL. The API and policy name are passed in SYSIN.

13.5.10 PTYQDKID

This PTYQDKID COBOL program queries the Data KEYID and writes the fetched value in the CXI-Reason-Code. It uses any of the stateful interfaces. The API and data element are accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYJQDK is the compile JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.

b. The PTYRQDK is the runtime JCL. The API and data element are passed in SYSIN.

13.5.11 PTYVSAM

This PTYVSAM COBOL program protects the data in VSAM file and writes it to another VSAM file. It uses any of the stateless interfaces. The data element is accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYJVSM is the compile JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.

b. The PTYRVSM is the runtime JCL. The data is loaded from APDATA1 input file to INPVSAM file. The PTYVSAM protects the INPVSAM and writes it to the CIPVSAM file. Since the VSAM primary key is also part of the protected data, so while writing the protected data to CIPVSAM, the job can fail with `Status-Code=21`.

13.5.12 PTYVSAM1

This PTYVSAM1 COBOL program protects and unprotects the data in the VSAM file. The operations are done using the length preserving tokens and the VSAM primary key is not protected. The program uses any of the stateless interfaces. The API and data element are accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYJVSM1 is the compile JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.
b. The PTYRVSM1 is the runtime JCL. The data is loaded from the APDATA1 input file to INPVSAM file. The PTYVSAM1 protects the INPVSAM and writes it to the CIPVSAM file and unprotects the CIPVSAM and writes it to OUTVSAM.

13.5.13 PTYVSAM2

This PTYVSAM2 COBOL program protects and unprotects the data in the VSAM file. The operations are done using the non-length preserving tokens and the VSAM primary key is NOT protected. The program uses any of the stateless interfaces. The API and the data element are accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYJVSM1 is the compile JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.
b. The PTYRVSM2 is the runtime JCL. The data is loaded from the APDATA1 input file to the INPVSAM file. The PTYVSAM2 protects the INPVSAM and writes it to the CIPVSAM file and unprotects the CIPVSAM and writes it to OUTVSAM.

13.5.14 PTYVSAM3 and PTYVSAM4

The PTYVSAM3 program uses the CFUDATAF input file to be loaded in the input VSAM file. The PTYVSAM3 program protects and the PTYVSAM4 program unprotects the data in the VSAM file. In the PTYVSAM3 program, the data is read and protected in the INPUT procedure and after SORTing, the data is written to the CIPVSAM file. In the PTYVSAM4 program, the protected data is read and unprotected in the INPUT procedure and after SORTing, it is written to the OUTVSAM file. The operations are done using the length preserving tokens and the VSAM primary key is protected. The program uses any of the stateless interfaces. The API and data element are accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYJVSM1 is the compile JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6.
b. The PTYRVSM3 is the runtime JCL. The PTYVSAM3 protects the CFUDATAF input file and writes the protected data in the CIPVSAM. The PTYVSAM4 unprotects the CIPVSAM and writes the unprotected data in the OUTVSAM file.

13.5.15 PTYIMS1 and PTYIMS2

The PTYIMS1 uses the APDATAIM input file which when protected using the stateless interfaces, PTYPSLL or PTYPSLI, is loaded or inserted to the IMS database. In the PTYIMS2, data is selected from the IMS database and the unprotected data is

send to SYSOUT. The API and data element are accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYIMS2 should be in authorized PDS-E.

b. The PTYRIMS1 is the runtime JCL. The DBD and PSBs should be generated and available in the authorized PDS. The PTYIMS1 protects the APDATAIM input file and loads or inserts the protected data in the IMS database. The PTYIMS2 selects the protected data from the IMS database and writes the unprotected data on SYSOUT. There are various SET parameters which need to be provided before executing the JCL.

13.5.16 PTYIMS3 and PTYIMS4

The PTYIMS3 uses the APDATAIM input file which when protected using the stateful interfaces, PTYPCXL or PTYPCXI, is LOADed or INSERTed to the IMS database. In the PTYIMS4, the data is SELECTed from the IMS database and the unprotected data is send to SYSOUT. The API and data element are accepted from the DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name.

The following are the JCLs.

a. The PTYJIMS1 is the compile JCL. The default is COBOL V5R2, but the programs will also compile with COBOL v4 or v6. Specific IMS library as per the IMS version should be mentioned. The load modules for PTYIMS3 and PTYIMS4 should be in authorized PDS-E.

b. The PTYRIMS2 is the runtime JCL. The DBD and PSBs should be generated and available in the authorized PDS. The PTYIMS3 protects the APDATAIM input file and inserts the protected data in the IMS database. The PTYIMS4 selects the protected data from the IMS database and writes unprotected data on SYSOUT. There are various SET parameters which needs to be provided before executing the JCL.

13.5.17 PTYSLLDB

The PTYSLLDB program protects the data using PTYPSLL or PTYPSLI, inserts the protected data in DB2 samples, select the protected data from DB2 table and unprotect the table data into a file using PTYPSLL or PTYPSLI. Data-element is accepted from DDNAME SYSIN so that the program does not need to be recompiled just to change the data element name. Additional member PTYCRTBL contains the CREATE table query required in this COBOL program.

The following are the JCLs.

a. The PTYJSLDB is the compile JCL taking care of pre-compilation, compile, link-edit and bind the module. DBRMLIB and other specific DB libraries are to be mentioned as per the DB version. Default for compile is COBOL V5R2, but the programs will also compile with COBOL v4 or v6. The load modules for PTYSLLDB should be in PDS-E.

b. The PTYRSLDB is the runtime JCL. In addition to input and output files, DBRM library has to be mentioned. Data-element is put from SYSIN step. There are various SET parameters which need to be provided before executing the JCL.

13.5.18 PTYPSC1

The PTYPSC1 program shows how to use the alternate ID interface PTYPSLC.

PTYPSLC works like PTYPSLL for storage.

An additional parameter is provided after the external IV parameters. This parameter provides an alternate user ID that will be used as the policy user, in lieu of the signed-in user.

The following are the JCLS:

a. The PTYJSC1 is the compile JCL. The default is COBOL V5R2, but the programs will compile along with COBOL v4 or v6.

b. The PTYRSC1 is the runtime JCL. The Data Element is passed in SYSIN.

13.5.19 PTYPSEIV

This PTYPSEIV COBOL program is parameter driven. It uses the stateless interface by calling it dynamically instead of calling statically. The parameters are passed via the COBOL ACCEPT verb and DDNAME SYSIN.

Data element to be used, start position, length and the value of external IV are part of the parameters. Check the beginning of the Procedure Division to see the parameters and their order. The example runtime also shows the parameters. The PTYPSEIV uses the CFUDATAF input file.

Following are the the sample JCLs.

a. The PTYJEDL1 is the compile JCL. The default is COBOL V5R2, but the program will also compile with COBOL V4 or V6.

b. The PTYRSEIV is the runtime JCL. As shipped, the PTYPSLL is the interface program specified in the parameters. There is a print step for the input, cipher, and clear files. There is also a compare step for the input and clear files.

13.5.20 PTYPCEIV

This PTYPCEIV COBOL program is parameter driven. It uses the stateful interface by calling it dynamically instead of calling statically. The parameters are passed via the COBOL ACCEPT verb and DDNAME SYSIN.

Data element to be used, start position, length and the value of external IV are part of the parameters. Check the beginning of the Procedure Division to see the parameters and their order. The example runtime also shows the parameters. The PTYPCEIV uses the CFUDATAF input file.

Following are the the sample JCLs.

a. The PTYJEDL1 is the compile JCL. The default is COBOL V5R2, but the program will also compile with COBOL V4 or V6.

b. The PTYRCEIV is the runtime JCL. As shipped, the PTYPCXL is the interface program specified in the parameters. There is a print step for the input, cipher, and clear files. There is also a compare step for the input and clear files.

Appendix F

z/OS Database Protector Samples

This section contains the sample programs for performing various tasks using the z/OS Database Protector.

The following table describes the samples.

Table F-1: Sample Programs

No.	JCL	z/OS Database Protector	Short Description
PTYxxxxx.DB2SAMP			
1	DSN3SSGN		Sample SIGNON authorization Exit for RACF environment
2	FPRTEST	FIELDPROC	Sample query to protect table using FIELDPROC
3	FPRPRNT	FIELDPROC	JCL to print the table data to verify protected or unprotected data in tables
4	PTYETEST	EDITPROC	JCL to provide required parameters to PTYGEP to generate customized EDITPROC; 4 characters mentioned in EPR# will be appended to PTYE to create the EDITPROC EXIT name to be mentioned in CREATE table query in EPRTEST
5	PTYGEP	EDITPROC	JCL to execute PTYPGEP to build a customized EDITPROC exit program
6	PTYMDEP	EDITPROC	Defined EDITPROC map
7	EPRTEST	EDITPROC	Sample query to protect table using EDITPROC
8	EPRPRNT	EDITPROC	JCL to print the table data to verify protected or unprotected data in tables
9	FPRBTCH1	FIELDPROC	LOAD or UNLOAD data in tables protected or unprotected with FIELDPROC in BATCH
10	FPRBTCH2	FIELDPROC	INSERT or SELECT data in or from tables protected or unprotected with FIELDPROC in BATCH
11	FPRBTCH3	FIELDPROC	LOAD data from original table to FPR-protecetd table in BATCH
PTYxxxxx.UDFSAMP			
1	PTYSPEPS		Start the PEP server
2	PTYPPEPS		Stop the PEP server
3	UDFBTCH1		INSERT, SELECT, or UPDATE UDFs for encryption or decryption in BATCH
4	UDFTEST		Sample queries to test INSERT UDFs for encryption
5	UDFSLCT		Sample queries to test SELECT UDFs for decryption
6	UDFUPDT		Sample queries to test UPDATE UDFs for encryption
7	UDFTEST1		Sample queries to test INSERT UDFs for tokenization
8	UDFSLCT1		Sample queries to test SELECT UDFs for detokenization
9	UDFUPDT1		Sample queries to test UPDATE UDFs for tokenization

No.	JCL	z/OS Database Protector	Short Description
10	UDFPRNT		JCL to print the table data to verify protected or unprotected data in tables

Appendix G

z/OS File Protector Samples

This section contains the sample codes for performing various tasks using the z/OS File Protector.

15.1 Test data

There are three test data files included in the PTYnnnnn.FPSAMP library in XMIT format. Use the TSO RECEIVE command to reload the data.

1. The CFUDATAF contains approximately one thousand records with 16 character numeric data similar to credit card numbers, last names, and first names. This input data is for Fixed Blocked (FB) file formats.
2. The CFUDATAV contains approximately one thousand records with 16 character numeric data similar to credit card numbers, last names, and first names. This input data is for Variable Block (VB) file formats.
3. The CFUDATAU contains approximately one thousand records with 16 character numeric data similar to credit card numbers, last names, and first names. This input data is for Undefined (U) file formats.

15.2 Sample Programs

The following table describes the sample JCLs.

Table G-1: Sample Programs

No.	JCL	z/OS File Protector	Short Description
1	CONCAMEX	CSS - AMEX	Protect or unprotect the data in concatenated files
2	CONCAMLT	CSS - AMLT	Protect or unprotect the data in concatenated files
3	CONCCFU	CFU	Protect or unprotect the data in concatenated files
4	DEMCFU	CFU	Protect the data
5	DEMCFU2	CFU	Unprotect data, protected in job DEMCFU, and compare it with original data
6	DEMCSEX	CSS - AMEX	Protect the data
7	DEMCSEX2	CSS - AMEX	Unprotect data, protected in job DEMCSEX and compare it with original data
8	DEMCSLT	CSS - AMLT	Protect the data

No.	JCL	z/OS File Protector	Short Description
9	DEMCSLT2	CSS - AMLT	Unprotect data, protected in job DEMCSLT and compare it with original data
10	DEMFPCSS	CFU - CSS	Protect data with CFU and during unprotect, protect the unprotected data using CSS-AMLT
11	PTYRAMEX	CSS - AMEX	Protect, unprotect, or compare the data of FB, VB, and U format files
12	PTYRAMLT	CSS - AMLT	Protect, unprotect, or compare the data of FB, VB, and U format files
13	PTYRFILE	CFU	Protect, unprotect, or compare the data of FB, VB, and U format files
14	USSAMEX	CSS - AMEX	Protect or unprotect the data in UNIX System Service file
15	USSAMLT	CSS - AMLT	Protect or unprotect the data in UNIX System Service file
16	USSCFU	CFU	Protect or unprotect the data in UNIX System Service file

Appendix H

z/OS IMS Protector Samples

This section contains the following sample codes for performing various tasks with the IMS Protector for Mainframe.

16.1 Sample Code to Create DBD

The following sample code can be used to create DBD.

```
*********************************************************************
*                                                                   *
*          NAME:  PTYDBD                                           *
*                                                                   *
*   DESCRIPTION:  DBD GEN FOR LIBRARY DATABASE                      *
*   WITH COMPRESSION EXIT ROUTINE – PTYIC001                        *
*   HDAM WITH SEQ, FB                                               *
*********************************************************************
       PRINT   NOGEN
       DBD     NAME=PTYDBD,ACCESS=(HDAM),RMNAME=(DFSHDC40,1,5,200)
       DATASET DD1=INDD
       SEGM
NAME=LIBSEG,PARENT=0,BYTES=30,COMPRTN=(PTYIC001,KEY)
       FIELD   NAME=(BOOKID,SEQ,U),BYTES=5,START=1,TYPE=C
       FIELD   NAME=ISSDATE,BYTES=10,START=6,TYPE=C
       FIELD   NAME=RETDATE,BYTES=10,START=16,TYPE=C
       FIELD   NAME=STUID,BYTES=5,START=26,TYPE=C
       SEGM    NAME=BOOKSEG,PARENT=LIBSEG,BYTES=30
       FIELD   NAME=(BOOKID,SEQ,U),BYTES=5,START=1,TYPE=C
       FIELD   NAME=BOOKNAME,BYTES=10,START=6,TYPE=C
       FIELD   NAME=BOOKAUTH,BYTES=15,START=16,TYPE=C
       SEGM    NAME=TECHSEG,PARENT=BOOKSEG,BYTES=5
       FIELD   NAME=(TBOOKID,SEQ,U),BYTES=5,START=1,TYPE=C
       SEGM    NAME=NTECHSEG,PARENT=BOOKSEG,BYTES=5
       FIELD   NAME=(NTBOOKID,SEQ,U),BYTES=5,START=1,TYPE=C
       SEGM    NAME=STUSEG,PARENT=LIBSEG,BYTES=25
       FIELD   NAME=(STUID,SEQ,U),BYTES=5,START=1,TYPE=C
       FIELD   NAME=STUNAME,BYTES=15,START=6,TYPE=C
       FIELD   NAME=STUDIV,BYTES=5,START=16,TYPE=C
       DBDGEN
```

```
        FINISH
        END
```

16.2 Sample Code to Create PSB

This section contains the following sample codes to create PSB.

The following are the sample codes to create PSB:

- *Creating PSB with the LOAD Operation*
- *Creating PSB with the SELECT Operation*

16.2.1 Creating PSB with the LOAD Operation

The following sample code can be used to create PSB with the LOAD Operation.

```
PRINT   NOGEN
PCB     DBDNAME=PTYDBD,TYPE=DB,KEYLEN=50,PROCOPT=L
SENSEG  NAME=LIBSEG
SENSEG  NAME=BOOKSEG,PARENT=LIBSEG
SENSEG  NAME=TECHSEG,PARENT=BOOKSEG
SENSEG  NAME=NTECHSEG,PARENT=BOOKSEG
SENSEG  NAME=STUSEG,PARENT=LIBSEG
PSBGEN  PSBNAME=PTYPSBL,LANG=COBOL,CMPAT=YES
END
```

16.2.2 Creating PSB with the SELECT Operation

The following sample code can be used to create PSB with the SELECT Operation.

```
PRINT   NOGEN
PCB     DBDNAME=PTYDBD,TYPE=DB,KEYLEN=50,PROCOPT=G
SENSEG  NAME=LIBSEG
SENSEG  NAME=BOOKSEG,PARENT=LIBSEG
SENSEG  NAME=TECHSEG,PARENT=BOOKSEG
SENSEG  NAME=NTECHSEG,PARENT=BOOKSEG
SENSEG  NAME=STUSEG,PARENT=LIBSEG
PSBGEN  PSBNAME=PTYPSBS,LANG=COBOL,CMPAT=YES
END
```

16.3 JCL Sample to create Segment edit/compression exit routine

The following is a JCL sample to create Segment edit/compression exit routine.

```
//PTYITEST     JOB ,MSGCLASS=H,NOTIFY=&SYSUID
//*-------------------------------------------------------------
//* PTYITEST - GENERATE CUSTOMIZED EXIT ROUTINE, PTYITEST
//*-------------------------------------------------------------
//*
//PTYITEST EXEC PTYGIM,DBD#=C001,DATAELEM='CUSP_TRDES',COMPRESS='Y'
```

16.4 JCL Sample to run programs to LOAD/SELECT the IMS database

This section contains the following JCL sample codes to run programs to LOAD/SELECT the IMS database.

The following are the JCL sample codes to run programs to LOAD/SELECT IMS database:

- *JCL Sample to run a program to LOAD the IMS database*
- *JCL Sample to run a program to SELECT the IMS database*

16.4.1 JCL Sample to run a program to LOAD the IMS database

The following JCL sample code can be used to run a program to LOAD the IMS Database.

```
//RUNJCLI  JOB ,MSGCLASS=H,NOTIFY=&SYSUID
//*-------------------------------------------------------------------
//* RUN PGMINS - ENCRYPTION
//*-------------------------------------------------------------------
//*STEP01    EXEC    PGM=DFSRRC00,PARM=(DLI,PROGRAM,PSB NAME,,,)
//STEP01    EXEC    PGM=DFSRRC00,REGION=4M,PARM=(DLI,PTYPIM1,PTYPSBL)
//STEPLIB  DD DSN=IMS1210.SDFSRESL,DISP=SHR
//         DD DSN=IMS1210.USER.SDFSRESL,DISP=SHR
//         DD DSN=IMS1210.PGMLIB,DISP=SHR
//         DD DSN=SURUCHI.IMS1210.LOADLIB,DISP=SHR
//INPFILE  DD DSN=SURUCHI.IMS1210.INPDATA,DISP=SHR
//DFSRESLB DD DSN=IMS1210.SDFSRESL,DISP=SHR
//         DD DSN=IMS1210.USER.SDFSRESL,DISP=SHR
//IMS      DD DSN=SURUCHI.IMS1210.PSBLIB,DISP=SHR
//         DD DSN=SURUCHI.IMS1210.DBDLIB,DISP=SHR
//INDD     DD DSN=SURUCHI.IMS1210.ESDSINP,DISP=(OLD,KEEP)
//PROCLIB  DD DSN=IMS1210.PROCLIB,DISP=SHR
//DFSVSAMP  DD *
VSRBF=8192,10
VSRBF=4096,5
VSRBF=2048,5
VSRBF=1024,5
VSRBF=512,5
//IEFRDER  DD DSN=IMSLOG,DISP=(,KEEP),
//         SPACE=(605,(500,500),RLSE,,ROUND),
//          DCB=(RECFM=VB,BLKSIZE=4096,
//          LRECL=4092,BUFNO=2)
//SYSUDUMP DD SYSOUT=A,
//         DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605),
//         SPACE=(605,(500,500),RLSE,,ROUND)
//IMSMON   DD DUMMY
```

16.4.2 JCL Sample to run a program to SELECT the IMS database

The following JCL sample code can be used to run a program to SELECT the IMS Database.

```
//RUNJCLS  JOB ,MSGCLASS=H,NOTIFY=&SYSUID
//*-------------------------------------------------------------------
//* RUN PGMSEL - DECRYPTION
//*-------------------------------------------------------------------
//*STEP01    EXEC    PGM=DFSRRC00,PARM=(DLI,PROGRAM,PSB NAME,,,)
//STEP01    EXEC    PGM=DFSRRC00,REGION=4M,PARM=(DLI,PTYPIM2,PTYPSBS)
//STEPLIB  DD DSN=IMS1210.SDFSRESL,DISP=SHR
//         DD DSN=IMS1210.USER.SDFSRESL,DISP=SHR
//         DD DSN=IMS1210.PGMLIB,DISP=SHR
//         DD DSN=SURUCHI.IMS1210.LOADLIB,DISP=SHR
//INPFILE  DD DSN=SURUCHI.IMS1210.INPDATA,DISP=SHR
//OUTFILE  DD DSN=SURUCHI.IMS1210.OUTDATA,DISP=SHR
//DFSRESLB DD DSN=IMS1210.SDFSRESL,DISP=SHR
//         DD DSN=IMS1210.USER.SDFSRESL,DISP=SHR
//IMS      DD DSN=SURUCHI.IMS1210.PSBLIB,DISP=SHR
//         DD DSN=SURUCHI.IMS1210.DBDLIB,DISP=SHR
//RECON1   DD DSN=IMS1210.RECON1,DISP=SHR
//RECON2   DD DSN=IMS1210.RECON2,DISP=SHR
//INDD     DD DSN=SURUCHI.IMS1210.ESDSINP,DISP=SHR
//PROCLIB  DD DSN=IMS1210.PROCLIB,DISP=SHR
//DFSVSAMP  DD *
VSRBF=8192,10
VSRBF=4096,5
VSRBF=2048,5
VSRBF=1024,5
VSRBF=512,5
//IEFRDER  DD DSN=IMSLOG,DISP=(,KEEP),
//         SPACE=(605,(500,500),RLSE,,ROUND),
//         DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092,BUFNO=2)
//SYSUDUMP DD SYSOUT=A,
//         DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605),
```

```
//          SPACE=(605,(500,500),RLSE,,ROUND)
//IMSMON    DD DUMMY
```

Appendix I

Security Considerations

This section contains suggestions about how to use the external security manager of z/OS to provide additional security for the Protegrity z/OS protector software.

Protecting the common software libraries:

- The *PTYnnnnn.SPTYAUTH* and *PTYnnnnn.SPTYAUTX* libraries only need to be readable by the started task shown as PTYCSRV in the *PTYnnnnn.SPTYSAMP* library. None of the members are used by any other Protegrity task and normal user applications do not need access to these libraries.
- The *PTYnnnnn.SPTYSAMP* library contains samples only. The member *PTYPARM* contains the samples that are the startup parameter for the *PTYCSRV* task, but do not need to remain in the *PTYnnnnn.SPTYSAMP* library to be used by the *PTYCSRV*. The *SPTYPARM DD* statement points to the appropriate member. The member *PTY$ENV* contains the Open MVS environmental variables for the *SPTYSPEP* JCL and needs to be in a library that the started task can access.

Protecting the Application Protector libraries:

- The *PTYnnnnn.APLOAD* library needs to be available to all the applications that will be using the Protegrity Application Protector. The members can be in site common application library or remain in the Protegrity library. All the members can be link-edited with the application or loaded dynamically. The applications should need only the read access to this library.
- The *PTYnnnnn.APSAMP* library contains the source sample programs and *PTYnnnnn.APSCNTL* library contains the associated JCLs to compile and run them.Some sample data used by several members are also included in the XMIT format. Only the read access should be necessary. Most of the programs will run in batch or in CICS.
- The *PTYnnnnn.APSLOD* library contains the compiled versions of several sample programs.

Protecting the Database Protector libraries:

- The *PTYnnnnn.DB2AUTH* library contains the executables that should be protected. Only the *xxxxDBM1* task needs the read and execute access to this dataset. The *xxxx* is the DB2 region name. Only the maintainer needs the write access.
- The *PTYnnnnn.DB2LOAD* library contains the utility programs. The *PTYPGEP* is used to generate a customized EDITPROC program. The *PTYPSGN* is a DB2 SIGNON exit program. Only the jobs that might use these need the read access.
- The *PTYnnnnn.DB2SAMP* library contains the sample JCL and SQL programs. The *EPRxxxx* and *FPRxxxx* members are the samples that show how the EDITPROC and FIELDPROC programs work. The *PTYGEP* is the JCL to execute the *PTYPGEP* to build a customized EDITPROC exit program. The combination adds the data element name to the EDITPROC.
- The *PTYnnnnn.UDFxxxx* libraries are for the Protegrity DB2 User Defined Functions (UDFs) and only need to be available if the UDFs are to be used. If the UDFs are not going to be used, then these libraries can be deleted.
- The *PTYnnnnn.UDFSQL* library contains the *CREATE* and *DROP* SQL commands to define the UDFs for DB2. These commands only need to be available for the DB2 DBA installing or uninstalling the UDFs. The *CREATE* command needs to be customized before use, as documented.

- The *PTYnnnnn.UDFUDFX* library contains the UDF executables. This library needs to be in the *STEPLIB* concatenation of the WLM job(s) that will run the SQL calling the UDFs. Only the WLM job(s) need the read and execute authority.
- The *PTYnnnnn.UDFSAMP* library contains the sample jobs to start and stop the PEP server. If you have run the install jobs in the *PTYnnnnn.SPTYSAMP*, then you can use the customized jobs from that process in lieu of these jobs. The UDFs only need the PEP server started and do not need the Protegrity Cryptographic Server. The *UDFxxxx* members are the samples that show how to use the UDFs.

Protecting the File Protector libraries:

- The *PTYnnnnn.FPAUTH* library contains the Protegrity executables that need to go into the LPA. These routines are called from the OPEN SVC. The library needs to be authorized, and have read and execute authority for z/OS. No individual job needs read or execute authority for these routines. You can leave these routines in the Protegrity library or use your own LPA library.
- The *PTYnnnnn.FPLOAD* contains the *PTYPFILE* member and the *PTYPVER* member which is called by the *PTYPFILE*. These routines needs to be in a library, available to all of the jobs that will use the Cryptographic File Utility. This library does NOT need to be authorized. The *PTYPFILE* runs in user key.
- The *PTYnnnnn.FPSAMP* library contains the JCL and sample data to show how the Cryptographic File Utility and Cryptographic SubSystem work. The sample data is in XMIT format in the library.

Protecting the Shared Memory:

- The PEP server (job PTYSPEPS), will create multiple shared memories of different sizes that are used as virtual files. All are multiples of 1 MB even though some could be smaller. Having the size being a multiple of 1 MB allows various tasks to share the shared memory and not just have a copy of the shared memory. This reduces the overall use of memory. The executables are not stored in any of these shared memories, only data is stored so no execute authority permission should be allowed.
- The shared memory with a size of 8388608 bytes or 8 MB (in binary) is the logging shared memory. This shared memory needs to be readable and writable by the PEP server, the Cryptographic Services Manager task, and by any WLM tasks that will use Protegrity User Defined Functions (UDFs). It is suggested that a Protegrity group be created in the External Security Manager, and that this group have the read and write access to this shared memory.
- The other shared memories need to have the read and write permissions for the PEP server task, but only readable by the rest of the Protegrity group.

> **Note:**
> In the file names the "nnnnn" refers to the product version. Unless otherwise stated, all are suggestions, not requirements, and all pertain to all the versions of Protegrity z/OS software.