PKZIP®/Smartcrypt® for IBM i®

User's Guide

SZIU- V16R00M03

PKWARE, Inc.

PKWARE, Inc.
201 E. Pittsburgh Avenue, Suite 400
Milwaukee, WI 53204

Main office: 888-4PKWARE (888-475-9273)
Main Fax: 414-289-9789

North American Sales: 866-583-1795 or 414-289-9788
International Sales (EMEA): 44-(0)207-4702420
Sales Email: pksales@pkware.com
Support: 937-847-2687
Support: http://www.pkware.com/support/system-i
Web Site: http://www.pkware.com

This edition applies to the following PKWARE Inc. licensed programs:
Smartcrypt for IBM i (Version 16, Release 0, 2016)

SecureZIP for z/OS, PKZIP for z/OS, SecureZIP for IBM i®, PKZIP for IBM i, SecureZIP for UNIX, and SecureZIP for Windows are just a few of the members of the PKZIP family. PKWARE Inc. would like to thank all the individuals and companies—including our customers, resellers, distributors, and technology partners—who have helped make PKZIP the industry standard for trusted ZIP solutions. PKZIP enables our customers to efficiently and securely transmit and store information across systems of all sizes, ranging from desktops to mainframes.

==
THIRD PARTY NOTICES

The aforementioned PKWARE products are licensed to contain the following third party programs. Any reference to these licensed programs or other material belonging to a third party is not intended to state or imply that such programs or material are endorsed by PKWARE, Inc. and/or currently available for use.

7/12/2017

Contents

Preface

Smartcrypt for IBM i, like ***PKZIP for IBM i***, is a member of the ***PKWARE*** family of products providing high-performance data compression and data protection across multiple operating systems and platforms.

With this release, PKWARE is changing SecureZIP's name to Smartcrypt across all supported platforms. The new name emphasizes the strong encryption that enterprises need to protect their data wherever that data travels. ***Smartcrypt for IBM i*** can provide the same capabilities as SecureZIP 14.0 maintenance level 1. When your installation of Smartcrypt is configured to match your existing SecureZIP environment, and existing jobs will continue to run without modification.

PKZIP for IBM i provides data compression on the AS/400, iSeries, i5 and IBM i. ***PKZIP for IBM i*** Enterprise Edition additionally includes support for passphrase-based decryption of encrypted files, powered by trusted OpenSSL. Files created by ***PKZIP for IBM i*** use the widely-adopted ZIP format and can be accessed on all major platforms throughout the enterprise—from mainframe to PC.

Smartcrypt for IBM i provides data compression and data protection on the AS/400, iSeries,i5 and IBM i. ***Smartcrypt for IBM i*** delivers high-performance data compression and protects data with digital signatures and trusted OpenSSL encryption, either passphrase- or certificate-based, with key lengths of up to 256 bits. Like ***PKZIP for IBM i***, ***Smartcrypt for IBM i*** uses the widely-adopted ZIP format and creates files that can be accessed on all major platforms throughout the enterprise.

This manual also covers ***SecureZIP Partner for IBM i***. ***SecureZIP Partner*** is a special version of ***Smartcrypt for IBM i*** that provides a straightforward, secure way for an organization to exchange sensitive information with outside partners who perhaps do not have Smartcrypt.

SecureZIP Partner for IBM i differs from the full ***Smartcrypt for IBM i*** in that it only extracts archives *from*, and only creates and encrypts archives *for*, a SecureZIP Partner sponsor. Contact PKWARE for more information on ***SecureZIP Partner***.

About this Manual

This manual provides information to help a system administrator install and use ***PKZIP for IBM i*** or ***Smartcrypt for IBM i*** in an operational environment on supported IBM releases of IBM i. It is assumed that people using this manual have a good understanding of (Control Language) CL and dataset processing.

Conventions Used in this Manual

Throughout this manual, the following conventions are used:

- ***Smartcrypt^{i}*** is used as a shorthand to refer to both ***Smartcrypt for IBM i*** and ***PKZIP for IBM i***. Statements made about ***Smartcrypt^{i}*** apply to both products. Information given specifically for ***Smartcrypt for IBM i*** or ***PKZIP for IBM i*** applies specifically to that product.
- The terms *ZIP* and *UNZIP* are used to refer to the respective overall processes of operating on an archive.
- The term *PKZIP* is often used generically to refer to any of the underlying executable programs that process archives in ***PKZIP for IBM i*** and ***Smartcrypt for IBM i***. These include programs PKZIP and SECZIP, to *ZIP* archives, and programs PKUNZIP and SECUNZIP, to *UNZIP* them. *PKZIP* is also more narrowly used to refer to either the PKZIP or SECZIP program, and PKUNZIP is often used to refer to either the PKUNZIP or SECUNZIP program.
- The use of the `Courier` font indicates text that may be found in control language (CL), parameter controls, or printed output.
- The use of *italics* in a command line indicates a value that must be substituted by the user, for example, a data set name. Italics are also used in body text to quote command names and so forth or to indicate the title of a manual or other publication.
- The use of <angle brackets> in a command definition indicates a mandatory parameter.
- The use of [square brackets] in a command definition indicates an optional parameter.
- A vertical bar (|) in a command definition is used to separate mutually exclusive parameter options or modifiers.

Program examples may show either ***Smartcrypt for IBM i*** or ***PKZIP for IBM i*** constructs, for backward compatibility. In general, examples apply to both programs unless the examples appear in sections of the manual that relate exclusively to ***Smartcrypt*** features. Such sections are marked like this:

Requires Smartcrypt

Related Publications

Smartcrypt^{i} product manuals include:

- *PKZIP/Smartcrypt for IBM i System Administrator's Guide* - Provides detailed information to assist the system administrator with the installation and administrative requirements necessary to use ***Smartcrypt^{i}*** in an operational environment.
- *PKZIP/Smartcrypt for IBM i User's Guide* - Provides detailed information on the product set in OS/400, i5/OS and IBM i operating environments. Also provided is a general introduction to data compression, SECZIP

specific data compression, and an overview on how to use ***Smartcrypt**^{i}*, SECZIP control cards, and parameters.

- *PKZIP/Smartcrypt for IBM i Messages and Codes* - This provides information on the messages and codes that are displayed on the consoles, printed outputs, and associated terminals.

Related IBM Publications

IBM manuals relating to the ***Smartcrypt**^{i}* product include:

- **System Messages:** This manual documents messages issued by the IBM i operating system. The descriptions explain why the component issued the message, provide the actions of the operating system, and suggest responses by the applications programmer, system programmer, and/or operator.
- **OS/400 CL Programming (SC41-5721):** This manual provides a wide-range discussion of iSeries Advanced Series programming topics, including: Control language programming, iSeries Advanced Series programming concepts, objects and libraries, and message handling.
- **OS/400 CL Reference (SC41-5722 thru SC41-5726):** This manual may be used in the iSeries Information Center to find information on the following CL reference topics: OS/400 commands, OS/400 objects, command description format, command parts, command syntax, about syntax diagrams, CL character sets and values, object naming rules, expressions in CL commands, and command definition statements.
- **Integrated File System Introduction (SC41-5711):** This book provides an overview of the integrated file system includes these topics:
- What is the integrated file system?
- Why might you want to use it
- Integrated file system concepts and terminology
- Interfaces you can use to interact with the integrated file system
- APIs and techniques you can use to create programs that interact with the integrated file system
- Characteristics of individual file systems
- **File Management (SC41-5710):** This manual describes the data management portion of the Operating System/400 licensed program. Data management provides applications with access to input and output file data that is external to the application. There are several types of these input and output files, and each file type has its own characteristics. In addition, all of the file types share a common set of characteristics.
- **DDS Reference (RBAF-P000):** This manual contains detailed instructions for coding the data description specifications (DDS) for files that can be described externally. These files are the physical, logical, display, printer, and intersystem communications functions, hereafter referred to as ICF files. You can also reference “Data description specifications” on the **IBM i and System i Information Center.**

Related Information on the Internet

PKWARE, Inc.

www.pkware.com

IBM

- o **IBM i and System i Information Center** - http://publib.boulder.ibm.com/eserver/ibmi.html

National Institute of Standards and Technology

- o **Computer Security Resource Center -** http://csrc.ncsl.nist.gov
- o **Information on the AES development** - http://csrc.nist.gov/encryption/aes
- o **Information on Key Management** - http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html

OpenPGP Alliance

http://www.openpgp.org/

RFC 4880 OpenPGP Message Format

http://www.ietf.org/rfc/rfc4880.txt

Release Summary

New Features 16.0.0 B

Smartcrypt for IBM i Release 16.0.0B introduces the following new features and changes:

- A binary symmetric cipher key may now be provided for ZIP archive file protection through a special "HEXKEY:" form of the PASSWORD command.
- Improve PGP key selection from large keyrings
- Fast PGP key selection from keyring
- Resolve year 2038 problem
- All accumulated fixed issues.

New Features 16.0.0

Smartcrypt for IBM i Release 16.0.0 introduces the following new features and changes:

- ***Smartcrypt for IBM i*** is the designated upgrade product for the ***SecureZIP*** product line on IBM i. The Smartcrypt product line provides the full complement of the latest SecureZIP product capabilities.
- All accumulated fixed issues.

- Support OpenSSL 1.0.1j
- Messages and displays may have changed text (Smartcrypt replacing SecureZIP) along with 2016 Copyright update.
- Allow hashing algorithm assignments when signing archives.
- Support creation of archive using BZIP2 compression algorithm.
- A new Smartcrypt NSSCLASSIFY setting that enables SECRET and TOP SECRET classification associated with Suite B cryptographic algorithms as specified by the National Institute of Standards and Technology (NIST) for protecting National Security Systems (NSS). Suite B includes cryptographic algorithms for encryption, digital signature, and hashing.
- Support retry when Resource Busy is detected while creating archives.
- View Smartcrypt assets encryption keys within a ZIP archive
- Support DSA 2048 Key sizes with OpenPGP files.
- Tolerate KEYID '0x' prefix for Selection of OpenPGP keys

New Features 14.0.1

SecureZIP for IBM i Release 14.0.1 introduces the following new features and changes:

- All accumulated fixed issues.
- Support OpenSSL 1.0.1e
- Support zlib and Bzip2 compression methods for the extraction of input OpenPGP archives from other sources
- Support creation with encryption method AE-2
- Support creation of archive using BZIP2 compression algorithm
- Support DSA 2048 Key sizes for OpenPGP signing/authentication
- The ability to use OpenPGP keyrings held in ASCII Armor format, including both ASCII and EBCDIC character set representations
- Support for creation and extraction of archives based on the older OpenPGP standard (RFC 2440)
- Support processing PGP "Signed Message data" with Signature armor data
- Update out-of-range check for V7R2M0 support. See Message AQZ9207.

New Features 14.0

New features available with ***SecureZIP for IBM i*** Release 14.0 include:

- All accumulated fixed issues.
- Informational message AQZ9207 to inform when operating environment is out-of-range for this release.

- Support for creation and extraction of archives based on the OpenPGP standard (RFC 4880).
- Use of OpenPGP keyrings for SecureZIP archive encryption and decryption

New features available with ***PKZIP for IBM i*** Release 14.0 include:

- All accumulated fixed issues.
- Support for creation non-encrypted archives based on the OpenPGP standard (RFC 4880).
- Support for extraction of archives based on the OpenPGP standard (RFC 4880). Password decryption and non- encrypted only. With no authentication.

New Features 10.0.5

New features in ***SecureZIP***^{i} Release 10.0.5 include:

- All accumulated fixed issues.
- UNZIP processing adds support for native block-mode processing for z/OS MVS data sets in RECFM=F or RECFM=V (unspanned) formats.
- Support for IBM i V6R1M0 and V7R1M0.
- Improved performance when selecting IFS files in folders with 1000+ entries.
- Improved tape handling when writing archives to tape, including the option to create a Shadow Directory file on tape.
- Ability to extract archives directly from tape –View, test and extract archives directly from tape. Use the Shadow Directory file for efficiency.
- Support for the newer signing Sponsor Distribution Packages for SecureZIP Partner.
- Support AE-2 password decryption.
- Support decompressing z/OS files that were compressed using z/OS CMPSC hardware compression.
- Increase field length for INCLFILE/EXCLFILE parameters from 30 characters to 255 to provide larger path lengths for list files.
- Improved iPSRA exception handling for PKZIP and PKUNZIP.
- PKZIP's parameter STOREPATH default was changed to be *REL in place of *YES.

New Features 10.0

New features in ***SecureZIP***^{i} Release 10.0 include:

- All accumulated bug fixes
- Ability to utilize IBM's cryptographic APIs, which in some cases may provide better performance.

- Enhanced self-extraction support for strongly encrypted archives and large archive support on specified releases of AIX, HP/UX, Linux, Sun Solaris or Windows
- Stronger digital signature digests with SHA-256, SHA-384 and SHA-512, as specified in FIPS 180-2
- ***PKZIP for IBM i*** will perform passphrase-based decryption of SecureZIP archives and associated files
- Tolerates UTF-8 file names in archive
- Accepts multi-byte (notably, UTF-8) text in certificates
- Ability to support Adopt Authority for archive files in libraries (added with V9.0.1)
- Improved information feedback when an API error occurs
- Increased maximum spool files that can be selected
- Improved text translation performance
- Updated ERROPT parameter for PKZIP, to assist skipping files with errors or iPSRA exceptions.

Command Changes & Defaults 16.0

The following commands have changed since version 14.0. Review each command and parameter listed in the User Guide (UG) or Systems Administrator's Guide (SAG) below before activating ***Smartcrypt*** 16.0. If the default will affect your current process you could change the defaults to accommodate an easier upgrade by using the IBM CHGCMDDFT command.

PKZIP/PKZSPOOL

SIGNPOL()	The option to assign a "Signing Hash" was added. This specifies the hashing algorithm that is used to generate a digital signature. It applies to the active Signing files and archives during a ZIP run.
NSSRULES ()	New Parameter. The NSS rules parameter controls the enterprise settings for adhering to their NSS process. There are currently two option settings for NSSRULES.
ARCHTEXT ()	New default text for *DEFAULT is: "Smartcrypt for IBM i".

PKUNZIP

NSSRULES ()	New Parameter. The NSS rules parameter controls the enterprise settings for adhering to their NSS process. There are currently two option settings for NSSRULES.

PKCFGSEC

NSSRULES ()	New Parameter. Define base NSS rules to control the enterprise settings for adhering to their NSS process.

New Commands 14.0

PKPGPZ

A command similar to PKZIP used to create an OpenPGP formatted file.

PKPGPU

A command similar to PKUNZIP used to display contents or to extract an OpenPGP formatted file.

PKARMOR

PKARMOR is utility that can encode an OpenPGP binary formatted file to a Radix-64 format (ASCII Armor) or decode a Radix-64 to a binary file.

Command Changes & Defaults 14.0

The following commands have changed since version 10.0.5. Review each command and parameter listed in the User Guide (UG) or Systems Administration Guide (SAG) below before activating ***SecureZIP^i*** 14.0. If the default will affect your current process you could change the defaults to accommodate an easier upgrade by using the IBM CHGCMDDFT command.

PKZIP/PKZSPOOL

CVTTYPE ()	The default is changed from *SUFFIX to *NONE.
DIRNAMES ()	Deprecated. Parameter was never implemented.
DFTARCHREC ()	The default is changed from 132 bytes to 1024 bytes.
ENTPREC ()	New LookUp Type of *PGPDEF to support the use of an OpenPGP keyring for encryption.
ARCHTEXT ()	New default text for *DEFAULT is: 'SecureZIP for IBM i'.
COMPAT ()	Deprecated. No longer providing the ability to build extended data in V4 formats.
PGPDEF ()	New Parameter. To define public and/or private OpenPGP keyrings with a unique handle to be referenced in ENTPREC parameter with option *PGPKRF.

ADVCRYPT ()	Removed the second element of this parameter. In previous version it was either *NONE or *BSAFE to control the encryption API. Last actual use was in V9.

PKUNZIP

CVTTYPE ()	The default is changed from *SUFFIX to *NONE.
DFTDBRECLN ()	The default is changed from bytes132 bytes to 1024 bytes.
ENTPREC ()	New LookUp Type of *PGPDEF to support the use of an OpenPGP keyring for decryption.
PGPDEF ()	New Parameter. To define public and/or private OpenPGP keyrings with a unique handle to be referenced in ENTPREC parameter with option *PGPKRF.

PKCFGSEC

ENTPREC ()	New LookUp Type of *PGPKRF for Contingency Key that supports the use of an OpenPGP keyring for encryption.
PGPRULE ()	New Parameter. Define base security rules for OpenPGP and use of OpenPGP keyrings.
PGPKEYPUB ()	New Parameter. PGPKEYPUB provides the ability to define a global OpenPGP public keyring and handle that can be referenced with PKPGPZ ENTPREC() and PKPGPU AUTHCHK() commands without defining a PGPDEF public keyring for each run.
PGPKEYPVT()	New Parameter. PGPKEYPVT provides the ability to define a global OpenPGP private keyring and handle that can be referenced with PKPGPZ SIGNER() and PKPGPU ENTPREC() commands without defining a PGPDEF private keyring for each run.
PGPPREC ()	New Parameter. The OpenPGP contingency key parameter defines the enterprise or corporate defined recipient which should be included as a global or administrative access recipient when creating OpenPGP files.

PKQRYCDB

FTYPE ()	A new file type (*PGPKRF) to support reading and displaying the contents of an OpenPGP keyring stored in the IFS.

New Commands 10.0.5

None.

Command Changes & Defaults 10.0.5

The following commands have changed since version 10.0.0. Review each command and parameter listed below before activating ***SecureZIP****i* 10.0.5:

PKZIP

STOREPATH()	The option default was changed from *YES to *REL. Review Migration notes 10.0.5 for consideration when installing this version.
PKOVRTAPF()	Added new “Shadow Directory File” option that controls the creation of a shadow directory file when writing archives directly to tape. Default is to create “Shadow Directory File”.

PKUNZIP

TYPARCHFL()	*TAP is added as an option to specify that the archive is to be read directly from tape using a tape device file.
PKOVRTAPI()	New Parameter to control and override attributes when reading archives directly from tape.

PKCFGSEC

ADVCRYPT ()	The mode is no longer operational for PKWARE or OpenSSL. It still is in the command for older version compatibility. See FACENC parameter for encryption facilities.

Migration Notes 14.0

An historical list of progressive release migration notes is provided below. PKWARE highly recommends that consideration be made of ALL pertinent release changes in relation to the release being replaced. Please contact PKWARE Technical Support if you have questions relating to the applicability of any of these items.

- Release 14.0 introduces support for encryption and decryption utilizing RFC 4880 OpenPGP key rings.
- Release 14.0 introduces support for RFC 4880 OpenPGP archive formats. As part of input archive processing for EXTRACT, TEST and VIEW actions, changes to the archive detection sequence have been m from a previous version.ade. Be aware that a change in behavior may occur when archives containing non-standard header or trailer material are read. In addition, when an error is detected during the initial read of the archive, the error handling characteristics of the run may differ from prior releases.
- The Configuration File (PKCFG) format has change with new fields therefore the **PKCFGSEC command MUST be run** to set all environmental settings in place of copying the file from a previous version.

Migration Notes 10.0.5

Release 10.0.5 introduces the ability to create a "Shadow Directory File" as the default when writing archives to tape. Creating the Shadow Directory File will add another file on tape immediately after the archive and will affect the number of files written to tape and the file sequence numbers. If the tape sequence number control of the files written to tape is extremely important, either change the command's **PKOVRTAPF()** options or change its defaults in the command using CHGCMDDFT.

The STOREPATH default has changed from *YES to *REL. This has the potential to affect subsequent file extraction that targets a UNIX or Windows file system. The ZIP archive standards document (also referred to as 'APPNOTE') states that the file name must not start with a leading '/'. Having a leading '/' can pose a security risk by allowing users to inadvertently overwrite files in the root of the directory during extraction. If your current processing requirements are dependent upon having a default of *YES, use the CHGCMDDFT command to change the STOREPATH parameter default in PKZIP and PKZSPOOL commands.

New Commands 10.0

PKCRYRUN was added for ***SecureZIP for i5/OS*** under OS V5R3M0 and above. PKCRYRUN is a utility testing command that simulates running encryption and hashing using specified facilities.

Command Changes & Defaults 10.0

The following commands have changed since version 9.0. Review each command and parameter listed below before activating ***SecureZIP***^{i} 10.0:

PKZIP

TYPARCHFL()	*XDB added as an option to specify that archive is to be created or updated exclusively in the QSYS library file system.
ADVCRYPT ()	The mode is no longer operational for PKWARE or OpenSSL. It remains available in the command for older version compatibility. See FACILITY parameter for encryption facilities.
FACILITY ()	New parameter to control encryption and hashing facilities or which API to use.

PKUNZIP

TYPARCHFL()	*XDB added as an option to specify that archive is to be created or updated exclusively in the QSYS library file system.
FACILITY ()	New parameter to control encryption and hashing facilities or which API to use.

PKCFGSEC

ADVCRYPT ()	The mode is no longer operational for PKWARE or OpenSSL. It remains available in the command for older

	version compatibility. See FACENC parameter for encryption facilities.
FACENC ()	New parameter to control encryption facilities or which API to use.
FACHASH ()	New parameter to control hashing facilities or which API to use.

Migration Notes 10.0

Release 10 introduced newer forms of self-extractor programs (ref. SELFXTRACT parameter for details) which support ZIP64 processing and strong decryption. Although the older versions of the self-extractors are still available, their names have changed. Jobs coded with the previous names will include the newer form of the self-extraction programs in the archive.

New Features 9.0

New features in ***PKZIP for i5/OS*** and ***SecureZIP for i5/OS*** Release 9.0 include:

- 1Step2Tape Feature – The ability to create an archive directly to tape without any disk files
- ***SecureZIP*** now supports multiple contingency keys with the use of inlist for a type code
- Expanded maximum passphrase length from 200 to 260 alphanumeric characters

New Commands 9.0

There are no new commands for version 9.0.

Command Changes & Defaults 9.0

The following commands have changed since version 8.1. Each command and parameter listed below should be reviewed before activating ***SecureZIP***^{i} 9.0:

PKZIP

TYPARCHFL()	*TAPF is added as an option to specify that archive is to be written directly to tape.
MSGTYPE()	An option is added that controls the amount of copyright information displayed at startup. Default is *NORMAL.
PASSWORD()	The key word *INLIST in the first 7 bytes indicates that the contents following will be an inlist file description where the passphrase will be retrieved.
TAPFOVR()	The TAPFOVR command is used to control the attributes when creating an archive directly to tape.

PKUNZIP

TYPARCHFL()	*XDB added as an option to specify that archive is to be read exclusively in the QSYS library file system.
MSGTYPE()	An option is added that controls the amount of copyright information displayed at startup. Default is *NORMAL.
PASSWORD()	The key word *INLIST in the first 7 bytes indicates that the contents following will be an inlist file description where the passphrase will be retrieved.

New Products 8.2

- The following product has been added to the PKWARE SecureZIP suite for the i5/OS operating environment:
- SecureZIP Partner for i5/OS

New Features 8.2

New features in ***PKZIP for i5/OS*** and ***SecureZIP for i5/OS*** Release 8.2 include:

- New compression algorithms with various custom controls
- Significant performance improvements with new compression algorithms
- New ZIP64 signal constraint checks to avoid building large archives
- New default internal translation tables for EBCDIC to ADCII
- A separate input archive can be specified other than the archive file to created. This allows an inputted archive to be preserved
- A special key word *COPY for the FILES parameter has been added that allows a zip run that just copies files from another archive
- The ability to extract zSeries files created with RDW (EBCDIC variable length records)
- i5/OS PKWARE Save/Restore Application feature (iPSRA)

New Commands 8.2

There are no new commands for version 8.2.

Command Changes & Defaults 8.2

The following commands have changes since version 8.1. Each command and parameter listed below should be reviewed before activating ***SecureZIP^i*** 8.2:

PKZIP

ARCHIVE()	Two additional options added (1. ZIP64 check and 2. Optional Input archive name). Defaults are backward compatible.

COMPRESS()	Additional options have been added. Nine (9) new compression levels for Level and a new option for compression method (Deflate or Deflate64). Defaults are backward compatible.
FTRAN()	Default has changed to *ISO88591. See Upgrade/Migration notes #1.
TRAN()	Default has changed to *ISO88591. See Upgrade/Migration notes #1.
FILES()	Revise to accommodate save commands for the iPSRA.

PKUNZIP

FTRAN()	Default has changed to *ISO88591. See Upgrade/Migration notes #1.
TRAN()	Default has changed to *ISO88591. See Upgrade/Migration notes #1.
RSTIPSRA()	The iPSRA Restore command

User Help and Contact Information

For Licensing, please contact the Sales Division at 937-847-2374 or email PKSALES@PKWARE.COM.

For Technical Support assistance, please contact the Product Services Division at 937-847-2687 or visit the Support Web site.

Appendix F lists the types of information needed to resolve issues with the product.

1 Getting Started

Smartcryptⁱ is a broad, flexible product on the IBM i platform, allowing for compression and decompression of files. It is fully compliant with other PKZIP-compatible compression products running on other operating systems.

Because the PKZIP standard for text data storage is ASCII, ***Smartcrypt^i*** facilitates conversion between the ASCII and EBCDIC character sets. Therefore, compressed text files can be transferred between IBM mainframe environments and systems using the ASCII character sets, including UNIX, DOS, and ***Smartcrypt^i***.

In addition to PKZIP-format archive support, ***Smartcrypt^i*** can also produce and manipulate (GNU) GZIP-format archives, (See Chapter 12), and OpenPGP archives (See Chapter 13).

PKZIP and PKUNZIP Commands

Smartcrypt^i uses two main commands—PKZIP and PKUNZIP—to control its high-performance data compression functionality. The PKZIP command launches a utility that compresses files and places them in a ZIP format archive. PKUNZIP reverses this process: it decompresses data in a ZIP archive created by PKZIP or another file compression program and restores the files to their original form. Both commands are controlled by options that allow a variety of functions to be performed.

Multiple levels of processing control are available through the use of customized option modules, shared command lists, and individual job inputs. In addition to file selection, features such as compression levels and performance selections can be specified. Also, a 32-bit cyclic redundancy check (CRC) is a standard feature used to guarantee data integrity.

A ZIP archive is platform-independent; therefore, data compressed (*ZIPPED*) on one platform, for example, UNIX, can be decompressed (*UNZIPPED*) on another platform, for example, IBM i OS and MVS/ESA, by using a compatible version of PKUNZIP.

PKPGPZ and PKPGPU Commands

For OpenPGP files, ***Smartcrypt^i*** utilizes two main commands—PKPGPZ and PKPGPU—to control its high-performance data compression and encryption functionality. The PKPGPZ command launches a program that compresses/encrypts/signs a file and places it into an OpenPGP format archive. PKPGPU reverses this process by decompressing/decrypting/authenticating data in an OpenPGP archive created by PKPGPZ or another OpenPGP file created to the RFC 4880 standard and restores the file to its original form. Both commands are controlled by options that allow a variety of functions to be performed. You must activate the PKPGPZ and PKPGPU commands with the PKCFGSEC command before you can use them.

Basic Features of *PKZIP for IBM i and Smartcrypt for IBM i*

Smartcrypt**^i* is generally compatible with ***PKZIP 2.x, and as such, has the following features:

Compliance with compression programs on other platforms, including Windows, Linux, UNIX, DOS, ***Smartcrypt**^i*.

- User-selected compression ratios.
- Storage capability of 65,535 files within one ZIP Archive.
- Compression of files of up to 4 gigabytes.
- A maximum ZIP archive size of 4 gigabytes.
- Data integrity assurance using 32-bit CRC error detection.
- Translation of data to a system-independent format, thus providing easy file transfers within a mixed or varied file environment.

***Smartcrypt**^i* also offers a series of extended features, such as creation of GZIP archives, spool files support, large file support (files greater than 4 GB and files in archive exceeding 65,535), advanced encryption, and self-extracting archives.

Initializing the License

Evaluation Period

You may obtain a key from the Sales Division to use to generate an evaluation license that allows full use of the product for 30 days. Contact PKWARE anytime during this period to obtain licensing to use the product beyond the initial period.

You can reach the Sales Division at 937-847-2374 or email pksales@pkware.com.

For technical support, contact the Product Services Division at 937-847-2687 or online at the Support Web site.

When you receive the license control card information from PKWARE, you build the license data set using the Build License program. Running the INSTPKLIC command updates the LICENSE data set and reports the license status of ***Smartcrypt**^i* at your location.

Release Licensing

Each release of ***Smartcrypt**^i* requires that a new license key be obtained from Customer Service and that a new license record be generated. The new release will fail with AQZ9077 "License Keys have invalid version setting" if the license file is used from a previous release.

Show System Information

To report on the status of a license at your location, you can run the environment "WHATOSV" program by doing a program call:➔CALL WHATOSV. It will provide a report similar to:

```
PKWARE WHATOSV Current Operating Environment  Wed Jan 13 07:40:59 2016

PKWARE SecureZIP Partner Smartcrypt(R) for IBM i Version 14.0.1 (540) with build
date 2012/04/12
              Current Smartcrypt  Library is PKW14061S
IBM iSeries Type 9406, Model  MMA-5462
            Proc_Feature<7380>  Int_Feature<    >
Serial Number <010-7X8WT  >, PRC Group < P30>,  OS is at V6R1M0.
Installed Processors(16)  - Activated Processors(16) - Max IBM i Processors(16)
POD/CoD Installed - Activated(16) Enabled/Active(00/00) Temp(00/00)
                  - Pod Feature(5403)

LPAR Data: Total Number of LPARs(63)
Current Partition:    Shared, Uncapped, U0D07X8WT002002
   ID(0x0D, 13)    Logical Serial Number<107X8WTD  >
    Processors:    Current(02)   Min(01)   Max(02)   Shared(16)
    Proc Capacity: Current(0.25) Min(0.10) Max(2.00)
                   VP(02) Current Cap Shared(0.00) Uncapped Weight(1.28)

 License Information for Product ID <5761SS1>:
          Feature<5050> licensed type Users       Lmt=*NOMAX Cnt=0 Peak=0
          Feature<5051> licensed type Processors  Lmt=*NOMAX Cnt=16 Peak=2
Press ENTER to end terminal session.
```

The output of this report is what you will need to send to your reseller or PKWARE sales representative to obtain a DEMO code.

Note: The *Smartcrypt*^{i} **Library must be added to the library list prior to running this program.**

Please have the output of this report handy when speaking with your reseller or account rep. You will be expected to supply the following additional information:

- Company name
- Company contact
- Phone number
- Contact email

Applying a License Key or Authorization Code

Installing the PKZIP license activation keys is done by adding the licensing information obtained from PKWARE, Inc. into a source file member (one is provided with distribution library call PKZLICIN) and then running the install license program to activate.

By executing the INSTPKLIC command, the LICENSE dataset will be updated and a report will be produced that will reflect the state of ***Smartcrypt***^{i} at your location.

Trial activation is accomplished by first editing the member PKWARELIC and adding the company customer record and keys supplied by PKWARE, Inc. One way of editing the member would to use the following command with the correct library:

➔EDTF FILE(PKW14053S/PKZLICIN) MBR(PKWARELIC)

or

➔STRSEU SRCFILE(PKW14053S/PKZLICIN) SRCMBR(PKWARELIC)

Remember since this a source file member and you use the EDTF command that the data will start in column 13, because the source sequence number and date stamp is in the true columns 1 thru 12.

For example:

➔ EDTF FILE(PKW14053S/PKZLICIN) MBR(PKWARELIC)

```
 Edit File: PKW14053S/PKZLICIN(PKWARELIC)
 Record :       1   of       3 by   8           Column :   13     92 by  74
 Control :

CMD ..+....2....+....3....+....4....+....5....+....6....+....7....+....8....+.
     ************Beginning of data*************
    *LICENSED BY PKWARE, Inc   06/03/03
    55 A4CMD1NR 000014581 PKWARE Internal Demo Customer
    99 CMDOAXB1 20030703 0107X8WTP10
     ************End of Data********************



 F2=Save   F3=Save/Exit   F12=Exit   F15=Services   F16=Repeat find
```

Notice in this case the columns on the ruler shows column 13 for the first column of the license data.

For example:

➔ STRSEU SRCFILE(PKW14053S/PKZLICIN) SRCMBR(PKWARELIC) :

```
 Columns . . . :    1  71            Edit                    PKW14053S/PKZLICIN
 SEU==>                                                               PKWARELIC
 FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
        *************** Beginning of data *************************************
0001.00 *LICENSED BY PKWARE, Inc   06/03/03
0002.00 55 A4CMD1NR 000014581 PKWARE Internal Demo Customer
0003.00 99 CMDOAXB1 20030703 0107X8WTP10
        ****************** End of data ****************************************




 F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Cursor   F11=Toggle
 F16=Repeat find       F17=Repeat change          F24=More keys
```

Once you have typed or copied the license information provided by PKWARE, you will need to save these changes by pressing F3 and exit the edited member by pressing F3 again. Next, run the install program using the following command:

➔INSTPKLIC INFILE(*LIBL/PKZLICIN) INMBR(PKWARELIC) or prompt F4

```
                         Install Smartcrypt for IBM i License (INSTPKLIC)

 Type choices, press Enter.

 Type . . . . . . . . . . . . . .   *INSTALL      *INSTALL, *VIEW
 Input Control File . . . . . . .    PKZLICIN       Name, PKZLICIN
   Library name . . . . . . . . .      *LIBL        Name, *LIBL
 Control Member . . . . . . . . .   pkwarelic        Name, *FIRST




                                                                   Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

By executing the INSTPKLIC command, the LICENSE dataset will be updated and a report will be produced that will reflect the state of ***Smartcrypt^i*** at your location.

```
Smartcrypt(R) for IBM i Version 16.0  (540),  2016/02/15
Portions copyright (C) 1989-2016 PKWARE, Inc. All rights reserved.
PKZIP Reg. U.S. Pat. and Tm. Off. Patent No. 5,051,745; 7,793,099; 7,844,579;
 7,890,465; 7,895,434
Other patent applications pending.
Smartcrypt(R) is a trademark of PKWARE, Inc.
Machine ID = 01061B5F, Processor Group =  P05, OS=V5R3M0
Rec -  1 *LICENSED BY PKWARE   05/26/06 WSS
Rec -  2 *Smartcrypt with Enterprise License
Rec -  3 57 MR6CCP2B 000015319 PKWARE, INC.
Rec -  4 99 HH6QYPKK 20100626 01061B5FP05
Evaluation Edition being installed
Smartcrypt Module      - Evaluation set to expire in 31 days on 20070626
Compression           - Evaluation set to expire in 31 days on 20070626
Decompression         - Evaluation set to expire in 31 days on 20070626
GZIP                  - Evaluation set to expire in 31 days on 20070626
Enhanced Decryption   - Evaluation set to expire in 31 days on 20070626
Spool Files           - Evaluation set to expire in 31 days on 20070626
Large Files           - Evaluation set to expire in 31 days on 20070626
Self Extracting       - Evaluation set to expire in 31 days on 20070626
iPSRA Save/Restore    - Evaluation set to expire in 31 days on 20070626
TapeOut IO Handler    - Evaluation set to expire in 31 days on 20070626
License File PKW14053S/PKZLIC(PKZLIC) Updated successfully
```

Reporting the License

By using the INSTPKLIC TYPE(*VIEW) command, the current licensing settings will be displayed.

```
Smartcrypt(R) for IBM i Version 16.0   (540),  2010/05/26
Portions copyright (C) 1989-2016 PKWARE, Inc. All rights reserved.
PKZIP Reg. U.S. Pat. and Tm. Off. Patent No. 5,051,745; 7,793,099; 7,844,579;
 7,890,465; 7,895,434
Other patent applications pending.
Smartcrypt(R) is a trademark of PKWARE, Inc.
Machine ID = 01061B5F, Processor Group =  P05, OS=V5R3M0
*********************************************
A License Report requested on 0107X8WT  from CPU Serial#
16.0 Product Licensed to Customer # 000014581 -PKWARE Internal Demo Customer
*********************************************
Compression           -DEMO with 23 Days remaining (07/03/2016)
      Contact PKWARE, Inc. for Licensing
*********************************************
Decompression         -DEMO with 23 Days remaining (07/03/2016)
      Contact PKWARE, Inc. for Licensing
*********************************************
GZIP                  -DEMO with 23 Days remaining (07/03/2016)
      Contact PKWARE, Inc. for Licensing
*********************************************
IFS File Handlers     -DEMO with 23 Days remaining (07/03/2016)
      Contact PKWARE, Inc. for Licensing
*********************************************
Database File Handlers-DEMO with 23 Days remaining (07/03/2016)
      Contact PKWARE, Inc. for Licensing
```

```
*********************************************
Advanced Encryption   -DEMO with 23 Days remaining (07/03/2016)
      Contact PKWARE, Inc. for Licensing
*********************************************
Spool Files           -DEMO with 23 Days remaining (07/03/2016)
      Contact PKWARE, Inc. for Licensing
*********************************************
Self Extracting       -DEMO with 23 Days remaining (07/03/2016)
      Contact PKWARE, Inc. for Licensing
*********************************************
Press ENTER to end terminal session.
```

PKZIP and *Smartcrypt for IBM i* Grace Period

PKWARE recognizes that there may be periods where the licensing environment established by the customer is no longer valid. Circumstances such as disaster recovery processing or the installation or upgrade of new processors will affect the environment.

To accommodate the installation, ***Smartcrypt**^{i}* has a process that will allow you to continue to use the product for a grace period of seven days when the established licensing environment is no longer valid. Note that the user *must* have write authority on the license dataset to invoke the grace period. This authority is only required the first time PKZIP/PKUNZIP is run after a CPU change has occurred; it is not required after the grace period has been successfully invoked (this is one time per CPU).

During the grace period, error messages will be displayed on the job log and/or display/printout for each execution of ***Smartcrypt**^{i}*. At the end of the period, if the license is not updated, the product will no longer function for the new CPUs except to VIEW an archive. You must contact PKWARE at pkcustomerservice@pkware.com during the grace period to obtain licensing to allow extended use.

Invoking *PKZIP for IBM i* or *Smartcrypt for IBM i*

Five main commands control ***Smartcrypt**^{i}* functionality in the IBM i operating environments. The commands are:

- PKZIP - Launches ZIP compression utility
- PKUNZIP - Launches ZIP extraction utility
- PKPGPZ – Launches OpenPGP compression utility
- PKPGPU – Launches OpenPGP extraction utility
- PKZSPOOL - Launches compression utility for spool files

Each of the commands can be invoked interactively, submitted for a batch run, or used anywhere that an IBM i command can be issued.

Help panels for each command can be activated by using the F1 (help) key.

Differences from *PKZIP* or *Smartcrypt* on Other Platforms

This section covers the differences between ***Smartcrypt***^{i} and other versions, including versions that run on other operating systems or platforms. Most of the differences are due to the QSYS library file type system and the IBM i object-oriented base.

Attributes (non-extended)	Various MS/DOS options support the selection of files by file attributes such as hidden, read-only, and system. These attributes are not meaningful on the IBM i OS file system.
ANSI comments	Because IBM i OS does not support ANSI control codes, related options are not supported. When unzipping from an archive, the archive comment will be displayed, but ANSI control codes in this comment will not be masked out. This could cause attribute changes on the IBM i display.
Archive file date controls *PKZIP*	DOS options control whether the ZIP file date is updated or retained when altering the archive. Because the last used date on IBM i OS is not under program control or alterable by a command, these options are not supported.
Archive Comments *PKZIP*	DOS options allow editing of comments for individual files in an archive. This version supports editing of a file's text description, but is not recommended for batch running, or for a large number of files due to the interactive message responses required.
File naming differences	The files used in the QSYS library file system have their own naming style. Each file associated with a library file and members would be depicted as library/file(member). Usually, all file names are stored as open system file names with directories, ending with a file name. For a detailed description and techniques see Chapter 5.
HELP	***PKZIP for DOS™*** has options to display a list of commands. Because ***Smartcrypt***^{i} uses IBM i commands, the help system is built for each command and is activated by PF1 on each parameter.
Mixed Case File Names	When using the IFS (Integrated File System), the file names are case sensitive and act like other file systems (UNIX, DOS, Windows, etc.). When using the QSYS library file system, the file names are always in UPPER CASE. Occasionally, when trying to update and archive (or select from an archive), you may encounter a case sensitive search. Use PKUNZIP view to get the exact name stored. This would be appropriate when doing a PKZIP TYPE(*DELETE) where the selection file would need to match.

Use of SAVF Method

At this time, only physical files with attributes of PF-DTA, PF-SRC, and SAVF in the QSYS file system, stream files and directories in the IFS, and spool files can be processed by ***Smartcrypt***^{i}**.** Also, some special database functionality such as triggers, file constraints, alternate collating sequence, and large object fields, are not stored in the archives.

To overcome some of the restrictions listed above, ***Smartcrypt**^{i}* can compress and decompress SAVF. The objects to be compressed are saved to a SAVF using SAVOBJ or SAVLIB. The SAVF is then compressed to an archive using PKZIP. To restore the data, first use PKUNZIP to re-create the SAVF, and then use RSTOBJ or RSTLIB to restore objects from within the decompressed SAVF. SAVF are binary and only pertain to IBM i OS.

Another solution may be to utilize the IBM i PKWARE Save/Restore Application Feature (iPSRA) feature, where the same command can issued with the save API based on the command defined in the FILES parameter. See Chapter 6 on the iPSRA feature for details.

Data Compression

Because data compression techniques reduce file size, a compressed data file will use less storage space and can be transferred in a faster, more efficient manner. A file can be compressed (a ZIP candidate) to a compact size (ZIPPED file), and then to use the file again, it must be uncompressed or extracted to its original size (UNZIPPED file).

One easy data compression method eliminates repeating or redundant data by replacing it with representative information that will be used when restoring the data. An example of this data compression technique is the Run-Length Encoding method, which applies to redundant data where a repeating character (the run) is represented as a count or value (the length). The compressed form is the repeated character with its count.

Example: B 2 2 2 2 E H H H H H H H H H

Compressed: B *4 2 E *9 H

Note: The efficiency of this method is dependent upon the amount of redundancy in the data.

To perform a thorough compression operation, more advanced algorithms and enhanced techniques are required. ***Smartcrypt**^{i}* uses just such methods to achieve maximum results.

ZIP Archives

***Smartcrypt**^{i}* is capable of storing compressed data in ZIP archives. There is no limit to the number of archives you may create.

ZIP archive capability:

- ZIP archive refers to any valid ZIP-format file created by a ***PKZIP*** 4.x-compatible product.
- ZIP64 refers to ZIP archives that include the ZIP64 format that can handle more than 65,534 files and files that exceed 4 GB. (See “Large Files Considerations.”).
- Each standard archive can store up to 65,534 files.

- Files that are over 4 GB have to be archived with GZIP or by using the Large File Support.
- Each standard archive may contain up to 4 GB of data. ZIP64 is required for larger archives.

For each file in the archive, the following information is stored with the compressed data:

- File name.
- File directory date and time.
- File's initial CRC value (see Cyclic Redundancy Check).
- Method of compression used.
- ***Smartcrypt****i* version required for file extraction.
- File size, uncompressed.
- File size, compressed.

Some files may contain the following additional information:

- The version of ***Smartcrypt****i* that created the file.
- File attributes.
- Any comment about the file.
- Any comment about the archive.
- Platform specific attributes (see Cross Platform Compatibility).
- If encrypted and what method of encryption.

Cyclic Redundancy Check

Cyclic redundancy check (CRC) is a method used to verify the integrity of a data file after it is restored from a ZIP archive.

Before a file is compressed, a ***Smartcrypt****i* algorithm computes a 32 bit hexadecimal value for its data. The CRC value is stored in a file that is within the ZIP archive. When the data in the file is extracted, ***Smartcrypt****i* processes it again using the same algorithm to produce a second CRC value. Once the file is processed, the original CRC value is compared to the new CRC value to ensure that they match.

Note: If the data is the same as its previous state, the same CRC value will be produced. When the two CRC values are compared, and should the extracted value not match the stored, initial value, the integrity of the file is in question and ***Smartcrypt****i* reports the results. In this case, it is possible the data was corrupted within the ZIP Archive.

Encryption

Requires Smartcrypt

Smartcrypt for IBM i can encrypt data for security control and provide a passphrase lockout for extracting data. Various security levels are available, with multiple encryption algorithms. See Chapter 2 for a description of security features in ***Smartcrypt for IBM i***.

File Selection and Name Processing

This section discusses how file selection is performed for ZIP processing with ***Smartcrypt****ⁱ*. The primary commands used for ZIP processing are discussed here, along with some overview notes and known restrictions.

This section also discusses how files are selected within an IBM i environment. Remember, ZIP directory entries within a ZIP archive will be defined in a system-independent format, which is not IBM i compatible.

Note: Directory entries within a ZIP archive are actually in a format compatible with UNIX systems and have been translated into the ASCII character set. In addition, the dataset level separators are typically set as the forward slash ("/"), not the period (".") as in IBM i, although this can be controlled through command actions in ***Smartcrypt****ⁱ***.**

See Chapter 5 for further information on how ***Smartcrypt****ⁱ* handles file name interchanges between IBM i and common ZIP format.

Primary File Selection Inputs

Smartcrypt*ⁱ* will only process:

- IBM i objects of type FILE (only with attributes PF-SRC, PF-DTA, and SAVF).
- IFS stream files (*STMR) and IFS directories(*DIR).
- Spool files.

Other objects must first be unloaded into an IBM i save file (SAVF) before they can be processed by ***Smartcrypt****ⁱ* (see: Use of SAVF Method) or use the Save Applications data with the iPSRA feature. See Chapter 6.

The FILES parameter in both PKZIP and PKUNZIP specifies which files are to be processed for all files except spool files (SPLF have their own selection parameters). One or more names can be specified, and each name is in either IBM i OS QSYS format, or IFS format, depending on F2ZTYPE settings. An asterisk may be used at the end of the library name, file name, or member name to select names beginning with the prefix used. To select all members of a file, *ALL may be used. To select all files in a library *ALL may be used (as long as it is qualified by at least a library name), for example, FILES('mylib/*ALL'). If *ALL is specified without at least a qualifying library name, the specification is ignored and no files will be selected.

The ***Smartcrypt****ⁱ* QSYS file system expands a partial file specification in several ways to make file specification more convenient. Each file specification may consist of a file name; a library name; a file name and member name; a library name and file name; or a library name, file name, and member name. IBM i SAVF may also be selected, but because a *SAVF file does not contain members, a SAVF will not be selected if a member name was included in the file specification.

In the Integrated file system, each file specification may consist of a directory, a path of directories, a directory and file, or a path of directories and file.

The various combinations that may be used are shown below:

File Type	***File specification***	***Expanded As***	***Notes***
QSYS	library*/	library*/*all(*all)	Finds all files in libraries beginning with *library*.
	fileinlib	*LIBL/fileinlib(*ALL)	Searches library list for all files called *fileinlib*. If a matching file is found, all of its members will be selected. If a SAVF is found, it will be selected.
	fileinlib*(mem*)	*LIBL/fileinlib*(mem*)	Searches library list for all files beginning with *fileinlib*. If a matching file is found, members beginning with *mem** will be selected. If a SAVF is found, it will NOT be selected because the file specification includes a member name.
	library*/file*	library*/file*(*ALL)	Searches libraries that begin with *library prefix* and for files that begin with *file prefix*. If a matching file is found, all of its members will be selected. If a SAVF is found, it will be selected.
	library*/file*(memo*)	library*/file*(mem*)	Searches libraries that begin with library prefix and files that begin with file prefix. If a matching file is found, members beginning with mem prefix will be selected. If a SAVF is found, it will not be selected because the file specification includes a member name.
IFS	Dir/*	Dir/*all	Searches all files in path DIR.

File Type	***File specification***	***Expanded As***	***Notes***
Spool Files	N/A		Uses parameters: SPLFILE, SFUSER, SFQUEUE, SFFORM, SFUSRDTA, SFSTATUS, SFJOBNAM, and/or SPLNBR.
iPSRA	Full Save Command		SAV, SAVLIB, SAVOBJ, SAVCHGOBJ, or SAVDLO

Note: If parameter TYPE(*DELETE) is used, then the file name format for these names must be in MS/DOS format (that is, if CVTFLAG has not been used). See the FILES keyword. Files may also be excluded. See the EXCLUDE keyword.

The valid parameter values for the FILES keyword are as follows:

'file specification 1' 'file specification 2'...'file_specification nn'

This is the list of one or more file specifications, separated by spaces.

For example:

mylib/myfile(prf*)

mylib/*all(*all)

By default, ***Smartcrypt**ⁱ* does a match on files in the QSYS library system with no case sensitivity and in the IFS with case sensitivity. Some IFS file systems contain case sensitive file names. To force ***Smartcrypt**ⁱ* to perform non-case sensitive file name matching use TYPFL2ZP(*IFS2).

File Exclusion Inputs

Using similar file specification techniques as described above in the Primary file Selection Inputs section, ***Smartcrypt**ⁱ* can specify from one to many file patterns that will be used to exclude files that were selected with the FILE parameter. The files can be inputted into the command parameter EXCLUDE or into a text file that can be processed by parameter EXCLFILE.

Care should be taken when using wildcards excluding inputs to ensure that FILES and EXCLUDE parameters select the desired files.

Input ZIP Archive Files

During a FRESHEN or UPDATE request, files contained within the existing ZIP archive are added to a candidate list. Names stored previously are used to search the system files for viability (any file names not found in the system remain in the ZIP archive).

SPOOL File Selecting

The FILES parameter is not used to select spool files for compression, but instead uses its own selection parameters.

There are eight positional parameters that can be specified to select the spool files: the SPOOL FILE NAME (SPLFILE), the SPOOL FILE NUMBER (SPLNBR), the user that created the files (SFUSER), the OUTQ that the file is residing (SFQUEUE), the form type specified (SFFORM), the user data tag associated with the spool file (SFUSRDTA), the status of the spool file (SFSTATUS), or the specific job name/user name/job number (SFJOBNAM). Only files that meet all of the selection values will be selected.

If the parameter SFJOBNAM is coded, the job must exist and the parameter SFUSER will be ignored, since it is already part of the SFJOBNAM parameter.

Large Files Considerations

Large File Support Summary

The large file support feature known as ZIP64 throughout this manual was added to ***PKZIP for IBM i*** in release 5.6. This separately licensed feature of ***Smartcrypt****i* provides several enhancements relating to capacity, size, and performance. Some of the key features include:

- Processing support (ZIP and UNZIP) for Archives enabled with the standard ZIP64 formats from other platforms.
- An increased ZIP archive file capacity is raised from 65,534 to the theoretical limit of 4,294,967,295 files.
- An increased user file size handler, raised from 4 Gigabytes minus 1 byte (32 bit binary counter) to a theoretical limit of 9 Exabytes (64 bit binary counter).
- An increased support for ZIP archive sizes exceeding 4 Gigabytes (same as user file size limit).

The preceding values are given only as theoretical limits. In practice, there are reasonable limitations due to the availability of resources along with processing tolerances.

Note: 4 GB or Gigabyte is equal to 4,294,967,295 bytes. 9 EB or Exabyte is equal to 9,223,372,036,854,775,807 bytes.

Large File Support File Capacities

The original .ZIP file format has faithfully met the needs of computer users since it was introduced by PKWARE in 1989. As computer technology has advanced over time, storage capacities have increased dramatically. These increases make the numbers and sizes of files that seemed unimaginable ten years ago a reality today. To extend the utility of the .ZIP file format to meet these changing system needs, PKWARE extended the .ZIP file format to support more than 65,535 files per archive and archive sizes greater than 4 Gigabytes (GB). This is known as the ZIP64 format.

The specification for the .ZIP file format has been publicly available and distributed by PKWARE in a file called APPNOTE.TXT. This file documents the internal data structures and layout that define a .ZIP archive. The extensions introduced by PKWARE fully support all the features of your existing archives and newer versions of

PKZIP that supports these new extensions will continue to read all of your current archives.

Prior to the 5.6 version of ***PKZIP for IBM i,*** PKZIP on the IBM i OS was limited to storing no more than 65,534 files in a .ZIP archive, and a single .ZIP archive or files in archive could not be larger than 4 GB (4,294,967,295 bytes). The extended ZIP64 file format specification available since PKZIP v5.6 supports creating .ZIP archives containing over 4 billion files and with sizes larger than 9 quintillion bytes. These are only theoretical limits and most IBM i systems and other computer systems in common use today do not have enough storage capacity, CPU or available memory to create and store ZIP64 archives approaching these limits.

The practical limits imposed by a typical IBM i system in use today and configured with various memory sizes will support compressing up to approximately 265,000 files. Compressing this number of files can take a long time, not only for the compression process, but to manage the directories and properties of each of these files.

Your available system resources (processor speed, DASD, memory, and other processing) limit the performance you can expect from ***Smartcrypt***^{i} when processing large numbers of files or large archives. If you are compressing a large quantity of files on an IBM i with insufficient memory or other resources you can expect slow processing.

When compressing large files, it is a good idea to have your archives set up to be stored in the IFS rather than in a library/file. The overhead is much less when storing the archive in the IFS. It is even more important when updating or adding to an archive where the temporary archive will also be processed in the IFS.

Versions of ***PKZIP for IBM i*** prior to 5.6 will not recognize these features and will be unable to view or extract any files in your archives that are dependent on these ZIP64 features. Also, any ZIP compatible programs you may be using from other companies will not be able to access all of the contents of your large archives. They may report that an archive is too large, or they may incorrectly report that the archive has errors. To ensure access to data in your large archives, always use genuine PKZIP/Smartcrypt from PKWARE.

Cross Platform Compatibility

Smartcrypt^{i} was designed for cross-platform use and enables you to move data among different computer operating environments. Archives created with ***PKZIP/Smartcrypt for IBM i*** are compatible with ***PKZIP for MVS***, ***PKZIP/SecureZIP for z/OS, PKZIP/SecureZIP for i5/OS***, ***PKZIP for OS/400***, ***PKZIP/SecureZIP for UNIX***, ***PKZIP/SecureZIP for LINUX***, ***PKZIP for DOS***, and ***PKZIP/SecureZIP for Windows***. All of these products use the same ZIP archive file format and can work with each other's archives. As a result, data can be zipped on one platform—for example, UNIX—and unzipped onto another platform, such as IBM i. ***Smartcrypt***^{i} automatically converts the data between EBCDIC and ASCII, so files prepared on the host are readable on any PC or UNIX system.

The following table lists ZIP features supported on different platforms and the version of the ZIP file format Application Note where the features appear. In the table, *(EE)* refers to ***PKZIP for IBM i*** Enterprise Edition.

ZIP Feature	***ZIP AppNote Version***	***z/OS***	***OS400/iSeries/IBM i***
Default	1.0		
File represents a volume label	1.1	Not supported	Not supported
File represents a folder	2.0	Not supported	Not supported
Deflate compression	2.0	2.x	2.x
Traditional encryption	2.0	2.x	2.x
Deflate64 compression	2.1	8.2	8.2
DCL Implode compression	2.5	Not supported	Not supported
File is a patched data set	2.7	Not supported	Not supported
File uses ZIP64 size extensions	4.5	5.6	5.6
BZip2 compression	4.6	Not supported	Not supported
DES encryption	5.0	SecureZIP 8.0	SecureZIP 8.1
3DES encryption	5.0	SecureZIP 8.0	SecureZIP 8.1
RC2 encryption	5.0	Not supported	Not supported
RC4 encryption	5.0	SecureZIP 8.0	SecureZIP 8.1
AES encryption	5.1	PK5.5, SZ8.0	PK5.5, SZ8.1
		(SecureZIP only, starting with 8.0)	(SecureZIP only, starting with 8.1)
DES decryption	5.0	SZ8.0, PK8.2(EE)	SZ8.1, PK8.2(EE)
3DES decryption	5.0	SZ8.0, PK8.2(EE)	SZ8.1, PK8.2(EE)
RC4 decryption	5.0	SZ8.0, PK8.2(EE)	SZ8.1, PK8.2(EE)
AES decryption	5.1	PK5.5, SZ8.0	PK5.5, SZ8.1
Digital Signatures and Authentication	5.1	SecureZIP 8.1	SecureZIP 8.1
Certificate encryption using non-OAEP key wrapping	6.1	SecureZIP 8.0	SecureZIP 8.1
Central directory encryption (file name encryption)	6.2	SecureZIP 8.0	SecureZIP 8.1
Certificate decryption	6.1	SZ8.0, PK11.0	SZ8.1, PK10.0
Hardware-based encryption	n/a	SecureZIP 9.0	Not Supported
Hardware-based decryption	n/a	SZ9.0, PK9.0(EE-decrypt)	Not Supported
DSNTYPE retention	6.4	SZ11.1, PK11.1	Not Supported
z/OS 1.11 EAV PS-E	6.4	SZ11.1, PK11.1	Not Supported

SecureZIP version 14.0 adds support for the following OpenPGP features:

OpenPGP Feature	***z/OS***	***OS400/iSeries/IBM i***
Decryption of OpenPGP-based files	SZ14.0, PK14.0	SZ14.0, PK14.0
Creation of OpenPGP-based files	SZ14.0	SZ14.0
Encryption of ZIP archives using OpenPGP keys	SZ14.0	SZ14.0
Decryption of ZIP archives using OpenPGP keys	SZ14.0, PK14.0	SZ14.0, PK14.0
Sign OpenPGP files with OpenPGP keys	SZ14.0	SZ14.0
Authenticate files signed with OpenPGP keys	SZ14.0, PK14.0	SZ14.0, PK14.0
Include references to OpenPGP public and private keyrings in signed or encrypted archives	SZ14.0, PK14.0	SZ14.0, PK14.0

If you want to transfer data across platforms using any other ZIP-compatible product, you should check with the supplier first to confirm which versions of PKZIP it is compatible with.

For more information regarding data formats, see “Data Format - Text Records vs. Binary Records” in Chapter 3 for a discussion regarding special considerations when transferring files between different platform types.

Restrictions

Due to various IBM i processing characteristics, the following restrictions should be carefully reviewed to determine the best way to proceed when using ***Smartcrypt**^{i}*:

***Smartcrypt**^{i}* in the QSYS file system will only work with objects that have an object type of *FILE and an attribute of PF-DTA, PF-SRC, and SAVF. To process other objects such as *PGM, *CMD, etc., use the SAVF method (see “Use of SAVF Method” in Chapter 1).

***Smartcrypt**^{i}* in the integrated file system (IFS) will only work with stream files (*STRM) and directories (*DIR).

Special database functionality, such as triggers, file constraints, alternate collating sequence, and logical files are not stored in an archive. To maintain this functionality, use the SAVF method (see “Use of SAVF Method”).

Special database fields for large objects (LOB) are not supported. These fields include: character large objects (CLOBs), double-byte character large objects (DBCLOBs), and binary large objects (BLOBs). In cases where the database contains one of these types of fields, use the SAVF Method.

Note: When creating OpenPGP archives, only one file may be stored in the archive.

2 Introduction to Data Security

Requires Smartcrypt

This chapter details how ***Smartcrypt for IBM i*** can strongly encrypt data for security control and protection. Much of the reference information in this chapter derives from the National Institutes of Standards and Technology. The NIST Computer Security Resource Center web site, *http://csrc.ncsl.nist.gov/*, contains FAQ's and documentation relating to computer security along with the Federal Information Processing Standard (FIPS) documents. In addition, the PKWARE web site, *WWW.PKWARE.COM*, contains information relating to security in ***Smartcrypt for IBM i***.

The following sections describe encryption, authentication, types of algorithms in use, information about specific mandates requiring the use of secure data and how ***Smartcrypt for IBM i*** will secure that data.

Note: *PKZIP for IBM i* provides support for password-based encryption and decryption using a 96-bit "Standard" encryption algorithm that is supported by older ZIP-compatible utilities. *PKZIP for IBM i* Enterprise Edition supports the decryption of all password-based algorithms provided in *Smartcrypt for IBM i*. 96-bit encryption is not supported for OpenPGP archives.

Encryption

Encryption provides confidentiality for data. The data to be protected is called plaintext. Encryption transforms the plaintext data into an unreadable form, called ciphertext, using an encryption key. Decryption transforms the ciphertext back into plaintext using a decryption key. Several algorithms have been approved in FIPS for the encryption of general purpose data. Each of these algorithms is a symmetric key algorithm, where the encryption key is the same as the decryption key. In order to maintain the confidentiality of the data encrypted by a key, the key must be known only by the entities that are authorized to access the data. These symmetric key algorithms are commonly known as block cipher algorithms, because the encryption and decryption processes each operate on blocks (chunks) of data of a fixed size.

FIPS 46-3 and FIPS 197 have been approved for the encryption of general-purpose data. The protection of keys is discussed below under Key Management.

Smartcrypt for IBM i uses symmetric key algorithms when encrypting user data.

Authentication is the process of validating digital signatures that may be attached to files in an archive or to an archive's central directory.

Authentication is a separate operation from data encryption. Whereas encryption is concerned with preventing parties from accessing sensitive data (such as private medical or financial information), authentication confirms that information actually comes unchanged from the purported source.

Authenticating digitally signed data both verifies the signature and validates the signed data.

Data Integrity

Smartcrypt uses a cyclic redundancy check (CRC) to ensure that data is successfully transferred into and out of a ZIP archive. The CRC process creates a unique hash value "thumbprint" from the original data stream. The thumbprint is regenerated at the receiving end and compared with the hash of the source for equality. The thumbprint value is stored independently of the data stream and is used during UNZIP processing to complete validation of the data.

Smartcrypt extends the concept of the CRC in two ways for the purpose of providing a tamper-resistant container within the ZIP archive. First, more rigorous HASH algorithms (MD5 and SHA-1) are used (as specified by the PKCFGSEC command with the parameter SIGNPOL) in addition to the 32-bit CRC to accurately reflect the uniqueness of the data stream. Second, the hash value is encrypted in a digital signature using a private-key certificate for the purpose of tamper detection after file extraction.

Note: CRC checking only applies to ZIP archives, not OpenPGP archives.

For more information regarding SHA-1 (Secure Hash Algorithm), see FIPS PUB 180-1, describing the Secure Hash Standard, at http://www.itl.nist.gov/fipspubs/fip180-1.htm.

Smartcrypt for IBM i provides the parameter SIGNERS (*ALL,*FILE, *ARCHIVE), to initiate the creation of digital signatures within the ZIP archive. The AUTHCHK command is used to perform a tamper check operation using the digital signature and hash.

National Security Systems Classification Support (Suite B)

National Security Systems document classifications are supported at the SECRET and TOP SECRET levels associated with Suite B cryptographic algorithms as specified by the National Institute of Standards and Technology (NIST). In the context of ***Smartcrypt for IBM i*** operations, Suite B includes cryptographic algorithms for encryption, digital signature creation and authentication, and hashing.

Smartcrypt for IBM i provides the NSSCLASSIFY setting to enable cryptographic specification enforcement for the SECRET and TOP SECRET levels. It also provides the NSSCHECK command to verify that input archive files conform to the specifications associated with a designated classification.

A detailed list of supported specification attributes may be found in the NSSCLASSIFY command section later in this manual. The basic list of specifications includes:

- AES encipher/decipher operations
- ECDSA digital signature support (ECC keys) with appropriate hash algorithms

For additional information about NIST Suite B specifications, see https://www.nsa.gov/ia/programs/suiteb_cryptography/

Digital Signature Validation

Smartcrypt makes use of certificate-based encryption within the Public Key Infrastructure (PKI) to generate and validate digital signatures. PKI provides an authentication chain for certificates to guarantee that the signature was created by the purported source. ***Smartcrypt*** supports the certificate chain authentication process by including necessary identification information within the ZIP archive. Subsequently, the certificate(s) used for signing can be authenticated through a complete chain of trust. To complete the chain of trust, a *root* (or self-signed) certificate representing the certificate's issuing organization is installed on the authenticating system. This provides the receiving organization with the authority to declare how the final trust sequence should be treated. Signatures based on certificates from certificate authorities (CA) that are not authorized or trusted are declared as being *untrusted* by ***Smartcrypt***.

(OpenPGP keys do not have a "chain of trust", see chapter 13.)

Additional facets of validating a certificate's viability for use include a defined range of dates within which a certificate may be used and whether the certificate has been declared to have been revoked. Configurable Smartcrypt policies (EXPIRED and REVOKED attributes) provide support to ensure that the certificates involved in authentication also adhere to these restrictions.

Smartcrypt for IBM i provides a means to install and access the certificates necessary for signing and authentication. The AUTHCHK command, along with configured policy settings governs the type (archive directory or data files) and level of authentication that is to be performed.

Digital Signature Source Validation

A final step in completing the authentication process is to ensure that the archive and/or file data was sent from a particular source. Up to this point, using the previous two aspects of authentication, we are certain that the archive directory and/or files were signed with a private-key certificate that came from a trusted source (CA) and that the data stream has not been tampered with since it was placed into the ZIP archive. However, these steps alone do not guarantee that a different party under the same root/CA chain did not perform the signing operation.

Smartcrypt for IBM i provides an optional parameter in the AUTHCHK command to declare the specific party from whom the data is expected.

Example - Sign Files and Archive with Private Keys

Requires Smartcrypt

Create an archive and sign the files in the archive by two signers and then sign the archive directory. Note that signing requires the private key.

```
➔PKZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC03.zip')
 FILES('PKW14053S/$CONTACT') ADVCRYPT(AES256)
 TYPARCHFL(*IFS) TYPFL2ZP(*DB)
```

ENTPREC((*DB 'EM=PKTESTDB3@nowhere.com')
(*DB 'CN=PKWARE Test4'))
SIGNERS((*FILE *MBRSET 'pktestdb3.p12' (PKWARE))
(*ALL *MBRSET 'pktestdb4.p12' (PKWARE)))

```
Scanning files in *DB for match  ...
2  Encryption Recipients processed
Encryption Recipients List:
--CN=PKWARE Test3 EMail=PKTESTDB3@nowhere.com
--CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.com
2  File Signers processed
File Signers List:
--CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.com
--CN=PKWARE Test3 EMail=PKTESTDB3@nowhere.com
1  Archive Signer processed
Archive Signer List:
--CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.com
Found  1 matching files
Compressing PKW140XXS/$CONTACT($CONTACT) in TEXT mode
Add  PKW140XX.S/$CONTACT/$CONTACT  --  Deflating (80%)  encrypt(AES 256Key)
Smartcrypt Compressed 1 files in Archive /myroot/pkware/CStore/Testzips/TestC03.zip
Smartcrypt Completed Successfully
```

Example - Authenticate Signed Files and Archive

Requires Smartcrypt

When doing a basic view of the newly signed archive, notice that only the archive directory signatures are validated. To validate the signature of the files would require a TYPE(*TEST).

➔ **PKUNZIP   ARCHIVE('/myroot/pkware/CStore/Testzips/TestC03.zip')**
TYPE(*VIEW) TYPARCHFL(*IFS) TYPFL2ZP(*DB)
AUTHCHK((*ARCHIVE *MBRSET 'pktestdb4.crt')) AUTHPOL(*WARN (*ALL))

```
1  Archive Signer processed
Archive:  /myroot/pkware/CStore/Testzips/TestC03.zip   7053 bytes   1 file

  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
     5451 Defl:F     1702  69% 01-11-10 13:34 f091572d !PKW140XX.S/$CONTACT/$CONTACT
 --------         -------  ----                           -------
    5451             1702  69%                            1 file
Archive has been Digitally Signed.
Archive was signed by "PKWARE Test4" and verified
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

When the files in the archive are tested or extracted, the archive signature is validated first and then, after each file has been tested, the file's signatures are tested. If no AUTHCHK parameter is entered, all signatures are validated.

➔ **PKUNZIP   ARCHIVE('/myroot/pkware/CStore/Testzips/TestC03.zip')**
TYPE(*TEST) TYPARCHFL(*IFS) TYPFL2ZP(*DB)
ENTPREC((*DB 'CN=PKWARE Test3' 'PKWARE'))

```
1  Encryption Recipients processed
UNZIP Archive: /myroot/pkware/CStore/Testzips/TestC03.zip
Searching Archive /myroot/pkware/CStore/Testzips/TestC03.zip  for files to extract
Archive was signed by "PKWARE Test4" and verified
Testing: PKW140XX.S/$CONTACT/$CONTACT
File was signed by "PKWARE Test4" and verified
File was signed by "PKWARE Test3" and verified
PKW140XX.S/$CONTACT/$CONTACT tested OK
No errors detected in compressed data of /myroot/pkware/CStore/Testzips/TestC03.zip.
SecureUNZIP   Completed Successfully
```

Public-Key Infrastructure and Digital Certificates

Public-Key Infrastructure (PKI)

Use of digital certificates for encryption and digital signing relies on a combination of supporting elements known as a *public-key infrastructure* (PKI). These elements include software applications such as Smartcrypt that work with certificates and keys as well as underlying technologies and services.

The heart of PKI is a mechanism by which two cryptographic keys associated with a piece of data called a certificate are used for encryption/decryption and for digital signing and authentication. The keys look like long character strings but represent very large numbers. One of the keys is private and must be kept secure so that only its owner can use it. The other is a public key that may be freely distributed for anyone to use to encrypt data intended for the owner of the certificate or to authenticate signatures.

How the Keys Are Used

With encryption/decryption, a copy of the public key is used to encrypt data such that only the possessor of the private key can decrypt it. Thus anyone with the public key can encrypt for a recipient, and only the targeted recipient has the key with which to decrypt.

With digital signing and authentication, the owner of the certificate uses the private key to *sign* data, and anyone with access to a copy of the certificate containing the public key can authenticate the signature and be assured that the signed data really proceeds unchanged from the signer.

Authentication has one additional step. As an assurance that the signer is who he says he is—that the certificate with Bob's name on it is not fraudulent—the signer's certificate itself is signed by an issuing certificate authority (CA). The CA in effect vouches that Bob is who he says he is. The CA signature is authenticated using the public key of the CA certificate used. This CA certificate too may be signed, but at some point the *trust chain* stops with a self-signed *root* CA certificate that is simply trusted. The PKI provides for these several layers of end-user public key certificates, intermediate CA certificates, and root certificates, as well as for users' private keys.

X.509

X.509 is an International Telecommunication Union (ITU-T) standard for PKI. X.509 specifies, among other things, standard formats for public-key certificates. A public-key certificate consists of the public portion of an asymmetric cryptographic key (the public key), together with identity information, such as a person's name, all signed by a certificate authority. The CA essentially guarantees that the public key belongs to the named entity.

OpenPGP Keyrings

As defined by RFC 4880, paragraph 3.6 - Keyrings

> "A keyring is a collection of one or more keys in a file or database. Traditionally, a keyring is simply a sequential list of keys, but may be any suitable database. It is beyond the scope of this standard to discuss the details of keyrings or other databases."

OpenPGP keys are similar to X.509 keys in that they are represented as a public and private key pair and contain identify information such as a name and email address. They differ from X.509 in that there is no hierarchy of *trust* as there is with X.509 Certificate Authorities. Rather, there is a distributed web of trust.

Smartcrypt for IBM i supports OpenPGP keyring files in files stored on the IFS. These are separated into public (only) and secret (both secret and matching public) key sets.

See the PKOPGP01 member in QCLSRC for an example.

Digital Certificates

A digital certificate is a special message that contains a public key with identifying information about the owner, such as the owner's name and perhaps email address. An ordinary, end-user digital certificate is digitally signed by the CA that issued it to warrant that the CA issued the certificate and has received satisfactory documentation that the owner of the certificate is who he says he is. This warrant, from a trusted CA, enables the certificate to be used to support digital signing and authentication, and encryption of data uniquely for the owner of a certificate.

For example, Web servers frequently use digital certificates to authenticate the server to a user and create an encrypted communications session to protect transmitted secret information such as Personal Identification Numbers (PINs) and passphrases.

Similarly, an email message may be digitally signed, enabling the recipient of the message to authenticate its authorship and that it was not altered during transmission.

To use PKI technology in ***Smartcrypt for IBM i*** for encryption and to attach digital signatures, you must have a digital certificate.

Certificate Authority (CA)

A certificate authority (CA) is a company (usually) that, for a fee, will issue a public-key certificate. The CA signs the certificate to warrant that the CA issued the certificate and has received satisfactory documentation that the owner of the new certificate is who he says he is.

Private Key (X.509 or OpenPGP)

A *private key* is used to decrypt data encrypted with the associated public key and to attach digital signatures.

A private key must be accessible solely by the owner of the certificate because it represents that person and provides access to encrypted data intended only for the owner.

Smartcrypt for IBM i uses a private key maintained in X.509 PKCS#12 format and an OpenPGP private key from a secret keyring. In either case, the private key cannot be accessed unless a passphrase is entered for each Smartcrypt decryption or signing request.

Public Key (X.509 or OpenPGP)

A *public key* consists of the public portion of an asymmetric cryptographic key in a certificate that also contains identity information, such as the certificate owner's name.

The public key is used to authenticate digital signatures created with the private key and to encrypt files for the owner of the key's certificate.

Certificate Authority and Root Certificates

End entity certificates and their related keys are used for signing and authentication. They are created at the end of the trust hierarchy of certificate authorities. Each certificate is signed by its CA issuer and is identified in the "Issued By" field in the end certificate. In turn, a CA certificate can also be issued by a higher level CA. Such certificates are known as *intermediate* CA certificates. At the top of the issuing chain is a self-signed certificate known as the *root*.

Smartcrypt for IBM i uses public-key certificates in PKCS#7 format. The intermediate CA certificates are maintained independently from the ROOT certificates.

For examples using digital certificate encryption/decryption see User Encryption Examples.

Types of Encryption Algorithms

FIPS 46-3, Data Encryption Standard (DES)

The Federal Information Processing Standards (FIPS) specification 46-3 formerly specified the Data Encryption Standard (DES) algorithm for use in Federal government applications. In 2004, the specification was changed such that DES is no longer approved for Federal government applications.

Triple DES Algorithm (3DES)

Triple DES is a more recent algorithm related to DES. Triple DES is a method for encrypting data in 64-bit blocks using three 56-bit keys by combining three successive invocations of the DES algorithm.

ANSI X9.52 specifies seven modes of operation for 3DES and three keying options:

- the three keys may be identical (one key 3DES),
- the first and third key may be the same but different from the second key (two key 3DES), or
- all three keys may be different (three key 3DES).

One key 3DES is equivalent to DES under the same key; therefore, one key 3DES, like DES, has not been approved since 2004.

While two key 3DES (also known as 3DES-122) provides more security than one key 3DES (or DES), its use is deprecated for FIPS after 2010. Three key 3DES achieves the highest level of security for 3DES. NIST recommends the use of three different 56-bit keys in Triple DES for Federal Government sensitive/unclassified applications.

Smartcrypt for IBM i uses three key 3DES when Triple DES is selected as the data encryption algorithm.

FIPS-197, Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) encryption algorithm specified in FIPS 197 is the result of a multiyear, worldwide competition to develop a replacement algorithm for DES. The winning algorithm (originally known as Rijndael) was announced in 2000 and adopted in FIPS 197 in 2001.

The AES algorithm encrypts and decrypts data in 128-bit blocks, with three possible key sizes: 128, 192, or 256 bits. The nomenclature for the AES algorithm for the different key sizes is AES-x, where x is the size of the AES key. NIST considers all three AES key sizes adequate for Federal Government sensitive/unclassified applications.

Please see http://www.nist.gov/public_affairs/releases/g00-176.htm a press release recapping NIST's position

Smartcrypt for IBM i uses AES as the default encryption algorithm.

Comparison of the 3DES and AES Algorithms

Both the 3DES and AES algorithms are considered to be secure for the foreseeable future. Below are some points of comparison:

- 3DES builds on DES implementations and is readily available in many cryptographic products and protocols. The AES algorithm is new; although many implementers are quickly adding the algorithm to their products, and protocols are being modified to incorporate the algorithm, it may be several years before the AES algorithm is as pervasive as 3DES.
- The AES algorithm was designed to provide better performance (e.g., faster speed) than 3DES.
- Although the security of block cipher algorithms is difficult to quantify, the AES algorithm, at any of the key sizes, appears to provide greater security than 3DES. In particular, the best attack known against AES-128 is to try every possible 128-bit key (i.e., perform an exhaustive key search, also known as a brute force attack). By contrast, although three key 3DES has a 168-bit key, there is a "shortcut" attack on 3DES that is comparable, in the number of required operations, to performing an exhaustive key search on 112-bit keys. However, unlike exhaustive key search, this shortcut attack requires a lot of memory. Assuming that such shortcut attacks are not discovered for the AES algorithm, the uses of the AES algorithm may be more appropriate for the protection of high-risk or long-term data.
- The smallest AES key size is 128 bits; the recommended key size for 3DES is 168 bits. The smaller key size means that fewer resources are needed for the generation, exchange, and storage of key bits.

- The AES block size is 128 bits; the 3DES block size is 64 bits. For some constrained environments, the smaller block size may be preferred; however, the larger AES block size is more suitable for cryptographic applications, especially those requiring data authentication on large amounts of data.

See http://www.nist.gov/public_affairs/releases/g00-176.htm for a press release describing NIST's position on the two algorithms.

With a block cipher algorithm, the same plaintext block will always encrypt to the same ciphertext block whenever the same key is used. If the multiple blocks in a typical message were to be encrypted separately, an adversary could easily substitute individual blocks, possibly without detection. Furthermore, data patterns in the plaintext would be apparent in the ciphertext. Cryptographic modes of operation have been defined to alleviate these problems by combining the basic cryptographic algorithm with a feedback of the information derived from the cryptographic operation.

FIPS 81, DES Modes of Operation, defines four confidentiality (encryption) modes for the DES algorithm specified in FIPS 46-3: the Electronic Codebook (ECB) mode, the Cipher Block Chaining (CBC) mode, the Cipher Feedback (CFB) mode, and the Output Feedback (OFB) mode.

Smartcrypt for IBM i uses Cipher Block Chaining for data encryption.

RC4

The RC4 algorithm is a stream cipher designed by Rivest for RSA Security. It is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. Analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10^{100}. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software. Independent analysts have scrutinized the algorithm and it is considered secure.

RC4 is used for secure communications, as in the encryption of traffic to and from secure web sites using the SSL protocol.

CAST5 (aka CAST-128)

OpenPGP archive processing only

RFC 2144 defines a suite of CAST-128 algorithms with the potential of varying key lengths, up to 128 bits.

Smartcrypt for IBM i offers support for this algorithm in the 128-bit key form specified by OpenPGP RFC 4880.

IDEA

OpenPGP archive processing only

The International Data Encryption Algorithm (IDEA) is a block cipher designed by James Massey and Xuejia Lai. This algorithm was used in Pretty Good Privacy v2.0. IDEA is an optional algorithm in the OpenPGP standard, RFC 4880.

Smartcrypt for z/OS can decrypt files encrypted explicitly with this algorithm in the form specified by OpenPGP RFC 4880. Files using the PGP v2.6 format are not supported.

Key Management

The proper management of cryptographic keys is essential to the effective use of cryptography for security. Keys are analogous to the combination of a safe. If the combination becomes known to an adversary, the strongest safe provides no security against penetration. Similarly, poor key management may easily compromise strong algorithms. Ultimately, the security of information protected by cryptography directly depends on the strength of the keys, the effectiveness of mechanisms and protocols associated with keys, and the protection afforded the keys.

Cryptography can be rendered ineffective by the use of weak products, inappropriate algorithm pairing, poor physical security, and the use of weak protocols. All keys need to be protected against modification, and secret and private keys need to be protected against unauthorized disclosure. Key management provides the foundation for the secure generation, storage, distribution, and destruction of keys. Another role of key management is key maintenance, specifically, the update/replacement of keys.

Further information is available on key management at the NIST Computer Security Resource Center web site: http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html

Passphrases and PINS

FIPS 112, *Password Usage*, provides guidance on the generation and management of passphrases (passwords) that are used to authenticate the identity of a system user and, in some instances, to grant or deny access to private or shared data. This standard recognizes that passphrases are widely used in computer systems and networks for these purposes, although passphrases are not the only method of personal authentication, and the standard does not endorse the use of passphrases as the best method.

The passphrase used to encrypt a file with Smartcrypt may be from 1 to 260 characters in length. Different passphrases may be used for various files within a ZIP archive, although only one passphrase may be specified per run.

The passphrase is not stored in the ZIP archive and, as a result, care must be taken to keep passphrases secure and accessible by some other source.

Recipient Based Encryption

Passphrase-based encryption depends on both the sender and receiver knowing, and providing intellectual input (the passphrase) in clear text. The passphrase is used to derive a binary master session key for each decryption run. No key information is kept within the ZIP archive, so both parties must retain the passphrase in an external location.

Recipient-based encryption provides a means by which the master session key (MSK) information can be hidden, protected, and carried within the ZIP archive. This is done by using technique known as digital enveloping with public key encryption. The technique requires that the creating process have a copy of the recipient's public key digital certificate, which is used to protect and store the MSK. In addition, the receiving side must have a copy of the recipient's private key digital certificate. With these two pieces of information in place, there is no need for users to retain or recall a passphrase for decryption.

Integrity of Public and Private Keys

Public and private keys must be managed properly to ensure their integrity. The key owner is responsible for protecting private keys. The private signature key must be kept under the sole control of the owner to prevent its misuse. The integrity of the public key, by contrast, is established through a digital certificate issued by a certificate authority (CA) that cryptographically binds the individual’s identity to his or her public key. Binding the individual’s identity to the public key enables the key to be reliably used, for example, to authenticate signatures created with the corresponding private key.

PKI includes the ability to recover from situations where an individual’s private signature key is lost, stolen, compromised, or destroyed. This is done by revoking the digital certificate that contains the private signature key’s corresponding public key (discussed further below). The user then creates or is issued a new public/private signature key pair and receives a new digital certificate for the new public key.

OEM Cryptographic Extensions

In addition to the fully supported cross platform cryptographic services provided across the PKWARE product line, some additional extensions have been provided within ***Smartcrypt***^{i} to further assist with protected data exchanges.

GZIP 96-bit Passphrase Encryption/Decryption

Smartcrypt^{i} provides an extension to standard GZIP processing to support basic passphrase-based encryption and decryption support.

Operating characteristics are as follows:

- Passphrase value processing is required
- ADVCRYPT(ZIPSTD) is the only supported algorithm
- No certificate-based encryption/decryption processing is supported
- No digital signature support is available
- Encrypted GZIP files are supported between the following products:

o PKZIP for MVS

o PKZIP and SecureZIP for zSeries

o PKZIP and SecureZIP for z/OS

o PKZIP for OS/400

o PKZIP and SecureZIP for iSeries

o PKZIP and SecureZIP for IBM i

o PKZIP and SecureZIP for IBM i

AE-2 Passphrase Encryption/Decryption

- ***Smartcrypt**^i^* provides an extension to support the decryption of files held within a ZIP archive encrypted using the AE-2 algorithm used by other vendors, such as WinZip and 7-Zip.
- Operating decryption characteristics are as follows:
    - AE-2 with key strengths of 128, 192 or 256 are supported
    - AE-2 v1 and v2 implementations are supported
    - Passphrase value processing is required
    - No certificate-based encryption/decryption processing is provided
    - No digital signature support is available
- ***Smartcrypt**^i^* support creating AE-2 (AES256 only) encrypted files using the parameter/option of ADVCRYPT(AE_2).

Data Encryption

Smartcrypt for IBM i security functions include strong encryption tools using OpenSSL and IBM software cryptographic facilities. ***Smartcrypt for IBM i*** provides the option for passphrase encryption using DES, RC4, 3DES and AES.

Smartcrypt for IBM i uses a multi-layer key generation process, based on a user-specified passphrase of up to 260 characters, and/or a user's digital certificate, that creates a unique internal key for each file being processed. In addition, the same passphrase will result in a different system generated key for each file.

Smartcrypt for IBM i also implements the use of Cipher Block Chaining (CBC) to further enhance industry standard encryption algorithms. This feature ensures that each block of data is uniquely modified, further protecting the data from fraudulent access.

Smartcrypt for IBM i encryption is activated through the use of the PASSWORD and ENTPREC parameters. If a value is present for either setting, whether through commands or default settings, then encryption will be attempted in accordance with other settings (for example, -ADVCRYPT); however, if ADVCRYPT(*NONE) is specified, then encryption will be bypassed.

Operating System Levels

V5R1M0 or above is required to run certificate-based operations.

Windows Compatibility

When using OpenSSL AES encryption with recipients, there is a cross-system compatibility issue to be addressed by the user community. Windows operating systems running pre-Windows XP may experience a decryption problem depending on the state of the private-key certificate on the workstation. During the Windows certificate import process, a dialog check-box "Mark the private key as exportable" may be selected. If this option was not selected, then Windows will not allow an AES encrypted file to be decrypted unless the master session key was wrapped with 3DES.

The setting of the parameter is enterprise wide and is set using the PKCFGSEC command. When turned on, the MSK3DES flag is set in the NDH/DIB; indicating that the master session key information is protected with 3DES when recipients are specified.

PKZIP for Windows has a variance in processing for 6.0 and 7.x due to OAEP processing. PKZIP for Windows 5.0 through 6.0 used OAEP processing. However, that was found to be incompatible with Smartcards, so 6.1 and above began setting the NO_OAEP flag in the NDH/DIB flags and stopped creating OAEP encryption-mode files.

Smartcrypt for IBM i will always set NO_OAEP, therefore PKZIP for Windows 5.0 - 6.0 will not be able to read recipient-based files from the large platforms.

Smartcrypt for IBM i should be able to detect whether the NO_OAEP flag is set and successfully extract either. No change in logic is required within the Smartcrypt high-level code, but the low-level EVTCERTD code should handle the switch based on the flag.

What is File Name Encryption?

Someone who cannot decrypt the contents of an archive may still be able to infer sensitive information just from the unencrypted names of files. To prevent this, you can encrypt the names of files in addition to their contents. Encrypted file names can be viewed in the clear—that is, unencrypted—only when the archive is opened by an intended recipient, if the archive was encrypted using a recipient list, or by someone who has the passphrase, if the archive was encrypted using a passphrase.

Smartcrypt for IBM i encrypts file names using your current settings for (strong) encryption method and algorithm. File names can be encrypted using either strong passphrase encryption or a recipient list (or both). You must use one of the strong encryption methods: you cannot encrypt file names using traditional, ADVCRYPT(ZIPSTD), which uses a 96-bit key.

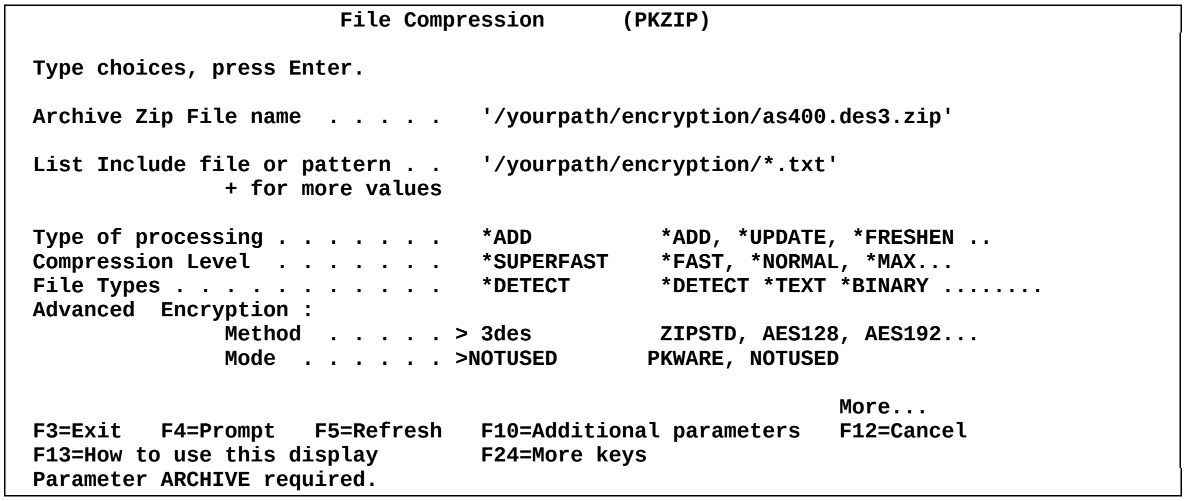
Encrypting names of files and folders in an archive encrypts and hides a good deal of other internal information about the archive as well. To encrypt file names, ***Smartcrypt for IBM i*** encrypts the archive's central directory, where virtually all such metadata about the archive is stored. Be aware, however, that archive comments are not encrypted even when you encrypt file names. Do not put sensitive information in an archive comment.

User Encryption Examples

Below are examples of how to invoke encryption processing using PKZIP commands.

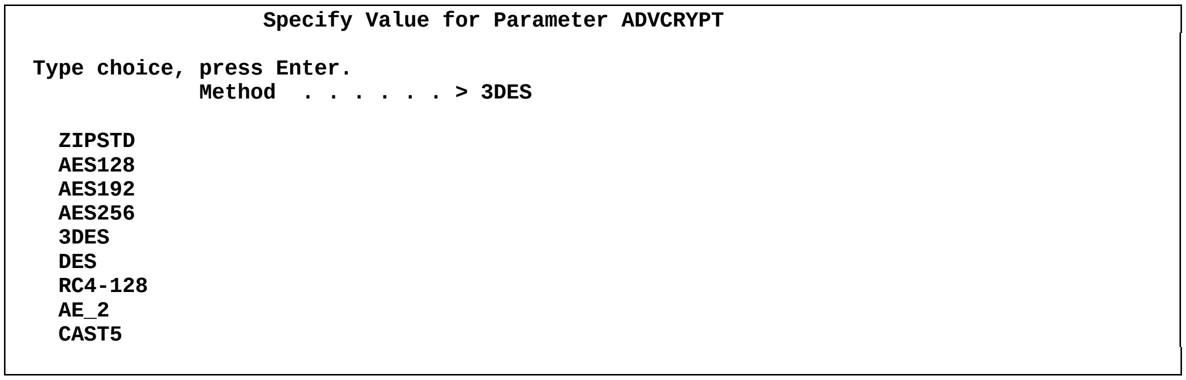
Zip Compress File(s) and Write to an Archive File

This is the main PKZIP compression screen. Here you specify the method and mode of encryption.



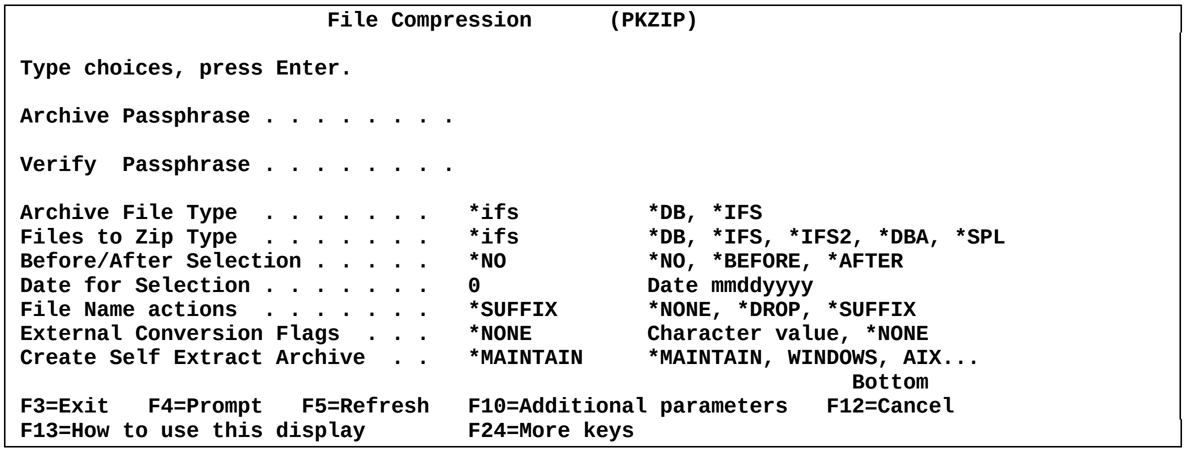
```
                          File Compression      (PKZIP)

 Type choices, press Enter.

 Archive Zip File name  . . . . .   '/yourpath/encryption/as400.des3.zip'

 List Include file or pattern . .   '/yourpath/encryption/*.txt'
               + for more values

 Type of processing . . . . . . .   *ADD           *ADD, *UPDATE, *FRESHEN ..
 Compression Level  . . . . . . .   *SUPERFAST     *FAST, *NORMAL, *MAX...
 File Types . . . . . . . . . . .   *DETECT        *DETECT *TEXT *BINARY ........
 Advanced  Encryption :
               Method  . . . . . > 3des          ZIPSTD, AES128, AES192...
               Mode  . . . . . . >NOTUSED       PKWARE, NOTUSED

                                                         More...
 F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
 F13=How to use this display        F24=More keys
 Parameter ARCHIVE required.
```

Placing the cursor on the "Method" and pressing F4 presents the next screen that allows you to select one of the encryption methods to use.



```
                    Specify Value for Parameter ADVCRYPT

 Type choice, press Enter.
             Method  . . . . . . > 3DES

   ZIPSTD
   AES128
   AES192
   AES256
   3DES
   DES
   RC4-128
   AE_2
   CAST5
```

When the next screen appears if you do not enter a passphrase no encryption processing is completed on the file(s) to be archived. If you desire encryption, you must enter the passphrase twice; once in the Archive Passphrase and again in the Verify Passphrase.



```
                          File Compression      (PKZIP)

 Type choices, press Enter.

 Archive Passphrase . . . . . . . .

 Verify  Passphrase . . . . . . . .

 Archive File Type  . . . . . . .   *ifs           *DB, *IFS
 Files to Zip Type  . . . . . . .   *ifs           *DB, *IFS, *IFS2, *DBA, *SPL
 Before/After Selection . . . . .   *NO            *NO, *BEFORE, *AFTER
 Date for Selection . . . . . . .   0              Date mmddyyyy
 File Name actions  . . . . . . .   *SUFFIX        *NONE, *DROP, *SUFFIX
 External Conversion Flags  . . .   *NONE          Character value, *NONE
 Create Self Extract Archive  . .   *MAINTAIN      *MAINTAIN, WINDOWS, AIX...
                                                               Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
 F13=How to use this display        F24=More keys
```

Following is the output of the PKZIP run.

```
Scanning files in *IFS for match  ...
Found  2 matching files
Compressing /yourpath/encryption/appnote.txt in BINARY mode
Add  /yourpath/encryption/appnote.txt  --  Deflating (69%)  encrypt(3DES)
Compressing /yourpath/encryption/readme.txt in BINARY mode
Add  /yourpath/encryption/readme.txt  --  Deflating (58%)  encrypt(3DES)
PKZIP Compressed 2 files in Archive /yourpath/encryption/as400.des3.zip
PKZIP Completed Successfully
```

Commands generated from the PKZIP screen using the retrieve key after the PKZIP run.

```
                                  Command Entry                          COSMOS
                                                            Request level:   4
 Previous commands and messages:

   (No previous commands or messages)



                                                                          Bottom
 Type command, press Enter.
 ===> PKZIP ARCHIVE('/yourpath/encryption/as400.des3.zip')
FILES('/yourpath/encryption/*.txt') ADVCRYPT(3DES) PASSWORD() VPASSWORD() TYPARCHFL(*IFS)
TYPF
```

Display the Contents of an Encrypted Archive File

When the files within an archive have strong encryption, the “!” (bang) character is placed in front of the file name to inform you that you must have the correct passphrase or private key certificate to view or extract the file. Files that are encrypted with the old standard (96 bit) password encryption will have the “+” character placed in front of the file name.

```
                        File Extraction     (PKUNZIP)

Type choices, press Enter.

Archive Zip File name  . . . . .   '/yourpath/encryption/as400.des3.zip'

List Include file or pattern . .   *ALL
               + for more values

Type of processing . . . . . . .   *VIEW         *VIEW, *EXTRACT, *NEWER...
File Types . . . . . . . . . . .   *DETECT       *DETECT *TEXT *BINARY ...


   Archive:  /yourpath/encryption/as400.des3.zip   33451 bytes   2 files
     Length  Method    Size  Ratio  Date   Time  CRC-32    Name
    -------- ------  ------- -----  ----   ----  ------    ----
       97182 Defl:F    30230  69% 01-30-04 13:16 223c2ea4 !/yourpath/encryption/
   appnote.txt
        5747 Defl:F     2710  53% 10-06-03 15:14 d193af9b !/yourpath/encryption/
   readme.txt
     --------         -------  ----                          -------
     102929           32940  68%                            2 files
   PKUNZIP extracted     0 files
   PKUNZIP Completed Successfully
```

Incorrect Passphrase Use

The following example shows the program's response to an incorrect passphrase. The error message indicates that the file(s) were skipped because of an incorrect passphrase and that PKUNZIP completed with errors.

```
    PKUNZIP Archive: /yourpath/encryption/as400.des.zip
    Archive Comment:"PKZIP for IBM i by PKWARE"
    Searching Archive /yourpath/encryption/as400.des.zip  for files to extract
    skipping: /yourpath/encryption/appnote.txt  incorrect passphrase
    skipping: /yourpath/encryption/readme.txt  incorrect passphrase
    Caution:  zero files tested in /yourpath/encryption/as400.des.zip.
    2 file(s) skipped because of incorrect passphrase
    PKUNZIP Completed with Errors
    Press ENTER to end terminal session.
```

Compress File with Public Digital Certificates

The first ZIP test will use both of the public certificates and 256-bit AES algorithm to encrypt and compress one file to an archive in the folder that was created earlier. This test will use the *MBRSET and *FILE types for the selection of the certificates.

**➔PKZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC01.zip')
FILES('PKW14053S/$CONTACT') ADVCRYPT(AES256)
TYPARCHFL(*IFS) TYPFL2ZP(*DB)
ENTPREC((*MBRSET pktestdb3.crt)
(*FILE '/myroot/pkware/CStore/Public/pktestdb4.crt'))**

```
Scanning files in *DB for match  ...
Total Recipients processed 2
Archive Recipient List:
CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.com
CN=PKWARE Test3 EMail=PKTESTDB3@nowhere.com
Found  1 matching files
Compressing PKW14053S/$CONTACT($CONTACT) in TEXT mode
Add  PKW14053.S/$CONTACT/$CONTACT  --  Deflating (80%)  encrypt(AES 256
Key)
Smartcrypt Compressed 1 files in Archive /myroot/pkware/CStore/Testzips/TestC0
1.zip
Smartcrypt Completed Successfully
```

The second ZIP test will use both of the public certificates and AES256 algorithm to encrypt and compress one file to an archive in the folder. This test will use the *DB with email and common name for the selection of the certificates.

**➔PKZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC02.zip')
FILES('PKW14053S/$CONTACT') ADVCRYPT(AES256)
TYPARCHFL(*IFS) TYPFL2ZP(*DB)
ENTPREC((*DB 'EM=PKTESTDB3@nowhere.com')
(*DB 'CN=PKWARE Test4'))**

```
Scanning files in *DB for match  ...
Total Recipients processed 2
Archive Recipient List:
CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.com
CN=PKWARE Test3 EMail=PKTESTDB3@nowhere.com
Found  1 matching files
Compressing PKW14053S/$CONTACT($CONTACT) in TEXT mode
Updating:PKW14053.S/$CONTACT/$CONTACT    Deflating (80%)  encrypt(AES 2
56Key)
Smartcrypt Compressed 1 files in Archive /myroot/pkware/CStore/Testzips/TestC0
2.zip
Smartcrypt Completed Successfully
```

Decrypting File with Private Key Certificates

Requires Smartcrypt

In order to decrypt the file you will need to provide at least one valid private certificate with the passphrase that matches a recipient on the archive.

```
➔PKUNZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC01.zip')
 TYPE(*TEST)
 TYPARCHFL(*IFS)
 ENTPREC((*DB 'CN=PKWARE Test4' ('PKWARE')))
```

Encryption Using LDAP Search for Recipients

Requires Smartcrypt

```
➔ PKZIP ARCHIVE('/yourpath/aVXXTest/test013.zip')
  FILES('/yourpath/aVXXTest/recp/Test cases.txt')
  TYPARCHFL(*IFS) TYPFL2ZP(*IFS) TYPLISTFL(*IFS)
  STOREPATH(*NO) ADVCRYPT(AES256)
  ENTPREC((*LDAP ‘EM=bill.Somebody@pkware.com' *N *RQD))
```

Displayed output from example.

```
Scanning files in *IFS for match  ...
Total Recipients processed 2
Archive Recipient List:
CN=PKWCADMIN EMail=none
CN=William Somebody EMail=bill.Somebody@pkware.com
Found  1 matching files
Compressing /yourpath/aVXXTest/recp/Test cases.txt in BINARY mode
Add  Test cases.txt  --  Deflating (81%)  encrypt(AES 256Key)
Smartcrypt Compressed 1 files in Archive /yourpath/aVXXTest/test013.zip
Smartcrypt Completed Successfully
```

3 ZIP Files

A ZIP archive is the storage facility for files that are compressed (or simply stored) using the ***Smartcrypt**^{i}* product. The basic archive can hold up to 65,535 files, which may have been compressed by up to 99% of their original size. Data integrity is validated by a cyclic redundancy check (CRC) to maintain integrity of the data from the compression through the extraction process. If the archive contains the ZIP64 archive format, the archive can support more than the 65,535 files and can be larger than 4 GB (see "Large Files Considerations" in Chapter 1).

In addition to the data, file attributes are retained, allowing extraction of the same file characteristics without the need of control card specifications. An archive can exist in three possible states during processing, described as "old archive," "temporary archive," and "new archive." An explanation of the functions of each of these is described in the sections below.

A ZIP archive is transferable between platforms. That is, files that are compressed by ***Smartcrypt**^{i}* on one platform may be extracted by ***Smartcrypt**^{i}* on a different platform, maintaining identical data.

This chapter describes the types of files used by ***Smartcrypt**^{i}* and provides a description of the way in which they are accessed by ***Smartcrypt**^{i}* ZIP archives.

Smartcrypt**^{i}* (by default) creates a new archives in the *DB file system as members of PF-DTA files with 132-byte records. The archive file is given a text field of "file created by ***Smartcrypt for IBM i" or "file created by ***PKZIP for IBM i***". The archive member is given a text field of "Member created by ***Smartcrypt for IBM i***" or "Member created by ***PKZIP for IBM i***". If you wish to create your own archive (perhaps to have a larger record size, for performance), then you can do so, but try to adhere to the following:

- When you create the file, do not create any members in it.
- After having created the file, change the MAXMBRS parameter for the file from 1 to *NOMAX.

A ZIP archive holds files internally in one of several formats, which are compatible with other platforms supported by ***PKZIP***. These formats are described here, and several commands are available for transforming files into one of these formats as they are compressed. You may specify in which format a file is stored using the FILETYPE(*BINARY) or FILETYPE(*TEXT) command parameters. IBM i OS SAVF are always stored as *BINARY type. If you do not specify FILETYPE(*BINARY) or (*TEXT), then the PKZIP and PKUNZIP programs both will default to FILETYPE(*DETECT). For more information, see FILETYPE(*DETECT).

“Old” ZIP Archive

Starting with ***SecureZIP^i*** Version 8.2, an optional input archive can be specified that can be a different name than the archive that will be created for an output archive file. If this is present, it is considered to be the “Old” ZIP archive. Otherwise the first ARCHIVE parameter is considered to be the “Old” ZIP archive.

The new input archive parameter (2nd option of ARCHIVE) allows the ability to preserve the input archive and create a new archive with a different name. This would allow the new archive to take on new attributes such as FNE or non FNE archive. The one requirement is that both archives must reside on the same file system, such as IFS or the QSYS Library file system.

Note: If you are updating an archive in the IFS, and it contains a hard link (the number of links is greater than 1), then the archive is not renamed at the end of the process. Instead the archive file is copied to overwrite the archive. This assures that the archive and all of the hard links are updated.

When there is not inputted archive, the 1st option of the ARCHIVE parameter for PKZIP is known as the old ZIP archive, except when the TYPE(*ADD) parameter is being used to create a new ZIP archive. The old ZIP archive may have been created by ***Smartcrypt^i*** during an earlier operation or may have been created by ***PKZIP*** on another platform and transferred from there. When a ZIP archive is being updated (or when PKUNZIP is extracting files from a ZIP archive), the necessary details are taken from the old ZIP archive. It should be noted that when ***Smartcrypt^i*** is updating a ZIP archive, it takes the necessary data from the old ZIP archive, merges it with any new data, and transfers it to a new ZIP archive (in a temporary member in the same IBM i file as the old archive). When all updating is completed, ***Smartcrypt^i*** deletes the old ZIP archive and then renames the new ZIP archive to the same name as the old ZIP archive. For this reason a file containing a ZIP archive should allow for at least one temporary member to be allocated. When ***Smartcrypt^i*** creates an archive file, it uses MAXMBRS(*NOMAX).

“Temporary” Archive File

A temporary archive file refers to an archive work in progress. ***Smartcrypt^i*** will always use a temporary archive file and its definition depends on the file system. If the file system type is IFS, then the temporary archive file will be in the same **directory** of the specified new archive. If the file system type is QSYS, the temporary archive file will become a **member** of the specified archive file. The temporary file or member will have a unique name PKnnnnnnnn (where nn represents an internal random number). When the file has been completed successfully, the temporary name will be renamed to the specified name in the ARCHIVE parameter. If this is a process in which an old archive is being updated, then (if successful) the old archive will be deleted before the rename. If a problem occurs, the temporary archive may stay with the temporary name. View the job log if this happens to determine the status of the archive.

“New” ZIP Archive

When the processing of the temporary dataset is finalized, ***Smartcrypt^i*** creates a new ZIPPED archive that is the modified “after” version of the old archive. The modified name of the old archive and specified allocation information is transferred automatically to the new archive after updating, and the old Archive is deleted. A new ZIP archive is created when an old ZIP archive is updated, or when a TYPE(*ADD) parameter (see Chapter 7) is used with ***Smartcrypt**^i^* where there is no old ZIP archive.

Self-Extracting Archive

The self-extracting programs are held as binary entities in the file PKZIPSFX of the ***Smartcrypt**^i^* library. The appropriate member is loaded and the executable data copied to the beginning of the Archive as a preamble when requested.

The resulting archive can still be processed by ***Smartcrypt**^i^* as a normal ZIP Archive.

When an input archive containing a self-extraction preamble is passed to ***Smartcrypt**^i^* for PKZIP processing and no value is supplied by SELFXTRACT, the default of *MAINTAIN will keep any preamble if one exist. If the parameter SELFXTRACT(*REMOVE) is supplied then the PREAMBLE is removed when writing the new archive.

A self-extracting archive can be created from an existing archive by using SELFXTRACT with a valid self-extractor. If the original archive contained a preamble, it will be removed and the newly specified preamble will be inserted.

When transferring a self-extracting archive to a target system, be sure to transfer the archive in binary format and adhere to requirements for executables in that environment. (For example, a Windows program should be saved with an application extension of EXE, and a UNIX file attribute should have executable authorization set via the UNIX chmod command).

For more details and usage notes see the PKZIP command parameter in Chapter 7.

In most cases, to include the path, use STOREPATH(*REL) or STOREPATH(*NOROOT). Do not USE STOREPATH(*YES) with self-extracting archives.

Use the ISRTPATH parameter to preset a path to store the files.

See Appendix G for information on options that are available when running self-extracting archives created by the various levels and types of self-extracting programs.

Data Format - Text Records vs. Binary Records

Binary data is stored in a ZIPPED archive in its original format. Binary data may be graphics or numbers that are already in “computer format.” Therefore, no translation is done, and EBCDIC will remain EBCDIC. The length of binary records in UNZIP processing is determined by the archive’s fixed-length records. ***Smartcrypt**^i^* will fill the available block automatically according to allocation specifications.

In the context of ZIPPED archives, a “text file” is one that is stored in the ASCII format. A text file contains records of data, each separated by a delimiter to signify the end of the record.

Note: An EBCDIC file containing text information (such as source code) can be stored in its original format by using BINARY, but it is not considered to be a “text” file within the ZIP architecture.

***Smartcrypt**ⁱ* uses the default line delimiter CR-LF (X’0D0A’) at the end of each text record. Text file members in the QSYS library file system use new line characters (NL=X’15’) internally. ***Smartcrypt**ⁱ* will handle the CR-LF and NL in both extraction and compressions automatically.

At the time of PKUNZIP file extraction, ***Smartcrypt**ⁱ* will convert text data from ASCII to EBCDIC by using a translation table. During installation, several translation tables are available, and the customization process will select one of the translation tables as a default. Additional translation tables may be created through the customizing procedure.

Situations may arise in unique platform interchanges, or when working with text files from other countries where the default text translation table is not adequate. Users may select any available translation table by using TRAN and FTRAN parameters.

***Smartcrypt**ⁱ* extracts text records stored in the ZIP archive by examining data for record delimiter and file terminator indicators. Using these indicators, ***Smartcrypt**ⁱ* aligns records in accordance with target file attributes.

Text files (such as program source code) are held within an archive using the ASCII character set for compatibility with other versions of ***Smartcrypt**ⁱ*. For these to be usable on IBM i OS, they must be converted to the IBM EBCDIC character set. Additionally, the carriage return and line feed characters must be removed before writing lines to a file because IBM i OS files are record-based and do not use control characters to separate records or lines. Text files usually have spaces at the end of a line. When using the text file handlers, ***Smartcrypt**ⁱ* has less data to read because the input/output routines remove trailing spaces and replace them with a new line character. This improves ***Smartcrypt**ⁱ* performance.

When extracting files from an archive, ***Smartcrypt**ⁱ* must know whether to perform text conversions. ***Smartcrypt**ⁱ* stores an indicator in the archive file’s local header defining if a file is binary or text-based. Because this indicator may be wrong in some circumstances, use the FILETYPE keyword to specify whether text conversions are required. When adding files to an archive, ***Smartcrypt**ⁱ* will flag the file according to the FILETYPE used.

***Smartcrypt**ⁱ* uses translation tables that should be suitable for most customers, but some users may wish to alter the tables. The procedure for changing the translation tables is discussed. If text files are only used on IBM i, then the FILETYPE(*EBCDIC) may be used. This uses IBM i files “as is” for the file (which are faster for text files), but does not translate the data to ASCII. This will provide a small improvement in performance.

Additionally, ***Smartcrypt**ⁱ* will translate each character in a text file from EBCDIC character format to ASCII character format by default. This is done using one of the two internal translation tables, which are named UKASCII and USASCII. It is recognized that these translation tables may not suffice for all countries or all situations, especially on those sites where text files are received from several different countries for processing into a single format. The source of the translation tables used by the PKZIP and PKUNZIP programs has been supplied, together with instructions for modifying the tables to create additional files (see Appendix D for details). This enables sites to modify the translation table as required.

In a case where FILETYPE is neither *TEXT nor *BINARY, *DETECT is the default mode. ***Smartcrypt***^{i} will read up to 64K of data from the input file and scan it for non-translatable text characters using the active text translation table. If any characters will not translate successfully using this method, the entire file will be treated as if *BINARY has been used.

Note: One exception to this is X'00' or the NULL terminator character, which is commonly used in C language. The NULL character will be allowed within the files. If file type is of a file in the archive is unknown whether it is text or binary, the user may use the TYPE(*VIEW) and VIEWOPT(*DETAIL) parameters to examine the file attributes.

File Attributes

Within each ZIP archive there are two different directories providing information about the files held in that archive. A local directory is included at the front of each file, with information pertaining to each file (for example: file size and date ZIPPED), and a central directory is located at the end of the ZIP archive. The central directory lists the complete contents of the ZIP archive and is the primary source of information for UNZIP processing.

Smartcrypt^{i} stores extended attributes about the file that can be useful in recreating the file during UNZIP processing. See the *System Administrator's Guide*.

PC Shared Drives Format

One common mistake made when extracting a text file to a shared drive folder in the IFS where the file will be used by a Windows application is to extract the file in text mode. Extracting a file as a TEXT file on the IBM i will cause PKUNZIP to translate the file to the EBCDIC format, since EBCDIC is the native IBM i format. The Windows application expects the file to be in ASCII, so therefore this file should be extracted using binary, since the files are stored in ASCII in the archive.

4 File Extraction Process

Extracting Files to the QSYS Library File System

Before extracting files to the QSYS library file system using TYPFL2ZP(*DB), among the questions to consider are:

- Does the file exist or will a new file be created?
- Did the file come from ***Smartcrypt^i*** or did it come from another platform?
- Are the files text type files from another platform where I need to know the record length?

If the file does not exist and if the file did not come from ***Smartcrypt^i***, you should provide a record length for the file with the parameter DFTDBRECLN. If the file is coming from ***Smartcrypt^i*** the record length will be in the extra data. If the file is from ***Smartcrypt^i*** and the parameter was DBSERVICE(*YES), the complete database definition from the extract data for database will be used to create the file.

If the file is to be created as text and the record length is too short then you will receive messages indicating the records are being truncated.

Two common parameters that are used to alter or guide the extraction process are the EXDIR and DROPPATH parameters. EXDIR provides the path library or library/file that the file will be extracted to when no library or path exist for the files in the archive. Of course, this is where the DROPPATH comes in to drop the first path or library with *LIB or to remove all paths in a name with *ALL.

For example, files in an archive might look like this:

```
Archive/#1:
My Document/myfiles/test/myheader.txt
My Document/myfiles/test/mydata.txt
My Document/myfiles/test/mytrailer.txt

Archive/#2:
    QGPL/QCLSRC/MYCL01
    QGPL/QCLSRC/MYCL02
    QGPL/QCLSRC/MYCL03
    QGPL/QCLSRC/MYCL04
```

In archive #1 let's assume that all three text files are of different records lengths. If we want to extract each with their own length, we would have to make three runs to create the files with different parameters. Or we could use CRTPF and create each of the files so the files would exist with the proper record length.

> **➔ PKUNZIP ARCHIVE('Archive/#1') FILES('My Document/myfiles/test/myheader.txt') TYPE(*EXTRACT) EXDIR('MYLIB/MYHEADER') DROPPATH(*ALL) DFTDBRECLN(50) CVTTYPE(*DROP)**

> **➔ PKUNZIP ARCHIVE('Archive/#1') FILES('My Document/myfiles/test/mydata.txt') TYPE(*EXTRACT) EXDIR('MYLIB/MYDATA') DROPPATH(*ALL) DFTDBRECLN(150) CVTTYPE(*DROP)**

> **➔ PKUNZIP ARCHIVE('Archive/#1') FILES('My Document/myfiles/test/mytrailer.txt') TYPE(*EXTRACT) EXDIR('MYLIB/MYTRAILER') DROPPATH(*ALL) DFTDBRECLN(20) CVTTYPE(*DROP)**

The commands above would create three files in library MYLIB, with all files having different record lengths.

Now suppose the files already exist with the names and record lengths. In this case, we could do all three files at once with:

> **➔ PKUNZIP ARCHIVE('Archive/#1') TYPE(*EXTRACT) EXDIR('MYLIB/?MBR') DROPPATH(*ALL) CVTTYPE(*DROP) OVERWRITE(*YES)**

The MBR will force each member name to also become the file name.

In archive #2, let's assume that we want to extract the CL source member and place them in a different library call MYNEWLIB. If the QCLSRC file does not exist and the archive was not built with DBSERVICE(*YES), then you would need to do a

> **➔ CRTSRCPF FILE(MYNEWLIB/QCLSRC)**

to have the file setup correctly for source files. If the QCLSRC file already exist in MYNEWLIB or the archive was built with DBSERVICE(*YES), then no special handling is required.

Authority Settings

When extracting files into the QSYS library file system, whether the files came from the AS/400 or another platform, the authorities are not taken from the archive, but from the user's current environment settings. The file's authority is not stored in the archive.

If a library is required to be created, PKUNZIP would create the library as if the current user was issuing a CRTLIB command. For standard settings, it might create the library with the following authority settings:

```
               Data     --Object Authorities--
User         Authority  Exist  Mgt  Alter  Ref
*PUBLIC      *RWX
USER         *RWX         X     X     X     X.
```

If the file does not exist, PKUNZIP will be required to create the file. If the file does exist no authorities are changed. If the file is created, the authority will be the same as if the user was issuing a CRTPF command in their environment. For most standard settings, it would create the file with the following authority settings:

```
             Data     --Object Authorities--
User        Authority  Exist  Mgt  Alter  Ref
*PUBLIC     *RWX
MYOWNER     *RWX         X     X     X     X
```

Extracting Files to the IFS

When extracting files to the Integrated File System with TYPFL2ZP(*IFS), record lengths are not a concern as they were in the QSYS library file system. The main considerations when extracting to the IFS is “what paths do you want for the file, or should the file be stored in EBCDIC or ASCII”.

Path Considerations

If the name of files in the archive, starts with ‘/’, then with no other changes this will be extracted to the root of the system with the first name in the path. This form of path name is called a fully qualified path. File names that starts with a ‘/’ can present a potential security hazard since the file will default to root structure of the IFS.

If the name does not start with a ‘/’, the item will be extracted to the paths based on the current directory (DSPCURDIR). This form of path name is called a relative path.

In both cases if the path(s) does not exist, the path(s) will be created with the attributes of the parent folder.

Changing the path(s)

In cases where the path that is stored with a name of the file in archive is not desired, then using the EXDIR and DROPPATH parameters should help guide the file to where it should be placed.

Using EXDIR, you can define the path of the file(s) that will be extracted. If you need to remove the path of the file in the archive, you can use DROPPATH(*ALL) to remove all the paths before extracting or you can use DROPPATH(*LIB) to remove only the first path name.

Again the coding of EXDIR follows the same rule with regards to fully qualified path or relative path.

File Type Considerations

When extracting a file, the decision to whether the contents of file should be stored in ASCII or EBCDIC needs to be made.

If the file is not a text file, it does not matter and should be stored as binary. If the file is text, and will be used by a PC program, chances are the data is expected to be in ASCII. Since the files are stored in the archive as ASCII, these files should be extracted as TYPEFILE(*BINARY). If the file is to be used by an AS/400 application or will be translated later, then chances are the file should be stored in EBCDIC. In this case use TYPEFILE(*TEXT) to extract the file in EBCDIC.

Authority Settings

If directories are required to be created during the extraction, the authority settings will be created according to the create directory definitions of the DTAAUT(*INDIR) and OBJAUT(*INDIR) parameters.

The authority for the directory being created is determined by the directory it is being created in. The directory immediately preceding the new directory determines the authority. A directory created in the root is assigned the public authority given to objects in the root directory. A directory created in QDLS for a folder defaults to *EXCLUDE for a first level folder. If created in the second level or greater, the authority of the previous level is used. The QOpenSys and root file systems use the parent directory's DTAAUT value.

The object authority is based on the authority for the directory where this directory is being created.

For IFS files, the access permissions flags of the file are captured. For example:

- S_IRUSR - Read permission for the file owner
- S_IWUSR - Write permission for the file owner
- S_IXUSR - Search permission (for a directory) or execute permission (for a file) for the file owner
- S_IRGRP - Read permission for the file's group
- S_IWGRP - Write permission for the file's group
- S_IXGRP - Search permission (for a directory) or execute permission (for a file) for the file's group
- S_IROTH - General read permission
- S_IWOTH - General write permission
- S_IXOTH - General search permission (for a directory) or general execute permission (for a file)

These access permission flags will be set for the owner that is running the PKUNZIP job and not the original owner.

Other user permissions from the parent folder will also be set for the file.

For example, the folder being extracted into has *PUBLIC as *EXCLUDE, the extracted file will also have *PUBLIC as *EXCLUDE.

Extracting zSeries Variable Length Records (RDW/ZDW)

In the zSeries, PKZIP can compress variable length records and store the files known as RDW or ZDW into an archive. The format of these records contains a 4 byte length (store in little Endian) followed by the record itself for that length. These records are stored in binary, therefore EBCDIC.

By using the TYPE(*DETECT), PKZIP will remove the record length before extracting the records. The ending format will differ depending on if the extraction is to a database file or to the IFS.

To extract to a fixed length database file, each record will be extracted and placed in the database forcing each record to be a fixed record in the database with no translation.

To extract to the IFS, each record will have a New Line (NL 0x15) character inserted at the end of each record. The records are still variable in length but with a separator. If the file was an object or load module, the results will be unpredictable.

If a file is extracted with TYPE(*TEXT), the results are unpredictable.

If a file is extracted with type(*BINARY), then the file is extract as is, including the 4 byte length field in front of each record.

Extracting zSeries Native IO Records

In the zSeries, PKZIP can compress records and store the files using Native IO record format method into an archive. The native blocks (with BDW and RDW controls) are saved in the data stream. A special ZIP directory attributed is used to signal that the data is in “BINARY Native Block I/O” format. The record data is stored in binary, therefore EBCDIC.

By using the TYPE(*DETECT), PKZIP will remove the block and record length before extracting the records. The ending format will differ depending on if the extraction is to a database file or to the IFS.

To extract to a fixed length database file, each record will be extracted and placed in the database forcing each record to be a fixed record in the database with no translation.

To extract to the IFS, each record will have a New Line (NL 0x15) character inserted at the end of each record. The records are still variable in length but with a separator. If the file was an object or load module, the results will be unpredictable.

If a file is extracted with TYPE(*TEXT), the results are unpredictable.

If a file is extracted with type(*BINARY), then the file is extract as is, including the any byte length fields in front of each record and blocks.

Extracting Spool Files

When extracting spool files with PKUNZIP, the attributes at the time of compression, will be preserved except for new spool file numbers. That will be generated. Parameter SPLUSRID is for the user ID on the new extracted spool file. If it is *DFT the original user ID will stay with the new spool file. Parameter SFQUEUE is for the OUTYQ and OUTQ library that the new extracted spool file will be placed. If *DFT is specified then the original OUTQ will be used to place the spool file.

Note on extracting spool files: To create or extract spool file with PKUNZIP, the user must have *USE authority to the API QSPCRTSP. The normal setting for the API QSPCRTSP is Authority PUBLIC(*EXCLUDE). The API authority is set this way so that system administrators can control the use of this API. This API has security implications because you can create spooled file from the data of another spooled file. To allow user to extract spool files change the API authority on a need basis.

When extracting a spool file with PKUNZIP, the new spooled file will be created with attributes based on values taken from the spooled file attributes when PKZIP archived the spool file. The spool file's file number, job, job user, job number, date, and time are controlled by IBM i operating system during the creation of a spool file.

The new spool file that is created by the PKUNZIP is spooled under one of two jobs and is dictated by IBM's create spool file API. The job is determined by the user-name field from the attributes. If the user name is the current user, it is a part of the user's job and is owned by the user profile that the job was started with. First the user profile for the user name must already exist. When using the user id override (parameter SPLUSRID), the spool file will be now belong to the override user.

If the ownership of the new spooled file is assigned to a different user by a different user profile name in the user-name field from the attributes, then the current user must have *SPLCTL authority to assign the spooled file to another user. When this is done, the new spooled file is by the user specified in the user name field or override parameter. The new spooled file is then part of a special system job (QPRTJOB) that is created for each user.

The new spooled file is placed on the output queue specified in the output queue name field from the original spool file attributes. If the parameter SFQUEUE is used it will override the attribute for the output queue.

In both cases, the spooled file name is the one contained in the spooled file attributes parameter. The spooled file number will be the next sequential one available for the job that the spooled file becomes a part of.

IBM i OS authority requirements when extracting spool files:

- *Special Authority* - *SPLCTL. This authority is needed if you are creating a spooled file for another user.
- Output Queue Authority -*USE
- Output Queue Library Authority - *EXECUTE
- Object QSPCRTSP API Authority -*USE

The following are several examples of results of extracting spool files:

Start with archiving the following spool files that were created with job MYJOB1 and user USER1:

```
File        File Nbr  Job         User        Number  Date      Time
QSYSPRT          397  MYJOB1      USER1       010893  12/11/09  13:35:29
QSYSPRT          398  MYJOB1      USER1       010893  12/11/09  13:36:09
QSYSPRT          399  MYJOB1      USER1       010893  12/11/09  13:36:09
```

Now extract with job MYJOB1 and user USER1 but now on a different day and job number:

```
File        File Nbr  Job         User        Number  Date      Time
QSYSPRT            2  MYJOB1      USER1       010927  12/12/09  09:57:42
QSYSPRT            3  MYJOB1      USER1       010927  12/12/09  09:57:42
QSYSPRT            4  MYJOB1      USER1       010927  12/12/09  09:57:42
```

Next extract with job MYJOB2 and user USER1 but now on a different day and job number:

```
File        File Nbr  Job         User        Number  Date      Time
QSYSPRT            2  MYJOB2      USER1       010928  12/12/09  09:59:06
QSYSPRT            3  MYJOB2      USER1       010928  12/12/09  09:59:06
QSYSPRT            4  MYJOB2      USER1       010928  12/12/09  09:59:06
```

Next, using the user, override with the SPLUSRID(WSS) and submit MYJOB1 with user USER1.

```
File        File Nbr  Job         User        Number  Date      Time
QSYSPRT           26  QPRTJOB     WSS         010118  12/12/09  10:02:50
QSYSPRT           27  QPRTJOB     WSS         010118  12/12/09  10:02:50
QSYSPRT           28  QPRTJOB     WSS         010118  12/12/09  10:02:50
```

Notice that the job was changed to QPRTJOB since the user being extracted was different than the user running the Job.

Next, signed on as user WSS, submit a job MYJOB11 with the job parameter USER profile specified for user USER1.

➔ **SBMJOB CMD(PKUNZIP ARCHIVE('atest/splftst/tst02') TYPE(*EXTRACT)) JOB(MYJOB11) USER(USER1)**

```
File        File Nbr  Job         User        Number  Date      Time
QSYSPRT            2  MYJOB11     USER1       010936  12/12/09  10:25:25
QSYSPRT            3  MYJOB11     USER1       010936  12/12/09  10:25:25
QSYSPRT            4  MYJOB11     USER1       010936  12/12/09  10:25:25
```

5 IBM i File Processing Support

***Smartcrypt**^{i}* can support files maintained in both the traditional QSYS library file system and in IFS (integrated file system) along with supporting spool files.

QSYS (Library File System)

The QSYS file system supports the IBM i library structure. This file system provides access to database files and all other IBM i object types that the library manages. On the IBM i system, each QSYS type file (also called a file object) has a description that details the file characteristics and how the data associated with the file is organized into records, and, in many cases, the fields associated for each record. Whenever a file is processed, the IBM i uses this description.

Of the objects in the library system, ***Smartcrypt**^{i}* will only process physical files that have an attribute type of PF-DTA (physical data files), PF-SRC (physical source file), or SAVF (save files).

QSYS files always exist in a library, and the PF-DTA and PF-SRC files (if data exist) will always have one too many members in the file. Therefore, PF-DTA and PF-SRC files have a name format of "library/file(member)." A SAVF (a special type of IBM i file for saving and restoring IBM i objects) does not have any members giving a file format of "library/file." Because SAVF types are handled in a special way, they are given additional consideration (see SAVF and use of SAVF method).

QSYS Summary

If the archive file is to be in the QSYS library system, set the Archive File Type parameter to TYPARCHFL(*DB). Even though *DB is working with archive files that are in a library file system, the IFS is utilized for performance and for large file support (ZIP64). If you need ***Smartcrypt**^{i}* to use the QSYS library system exclusively, for all processing—for example, to support OS400 features such as Adopt Authority—set the parameter to TYPARCHFL(*XDB).

If the file being compressed or extracted is in the QSYS library system, set parameter TYPFL2ZP(*DB).

If the list files (see Appendix C) are to be in the QSYS library system, set parameter TYPLISTFL(*DB).

Format Summary:

PF-DTA	LIBRARY/FILE(MEMBER)
PF-SRC	LIBRARY/FILE(MEMBER)
SAVF	LIBRARY/FILE

IFS (Integrated File System)

The Integrated File System is a part of IBM i which supports stream input/output and storage management similar to personal computer and UNIX operating systems, while providing an integrating structure over all information stored in the IBM i.

The key features of the Integrated File System are:

- Support for storing information in stream files that can contain long continuous strings of data. These strings of data might be, for example, the text of a document or the picture elements in a picture. The stream file support is designed for efficient use in client/server applications.
- A hierarchical directory structure that allows objects to be organized by specifying the path through the directories to an object for access to an object.
- A common view of stream files stored locally on IBM i, Integrated Netfinity Server for iSeries, or a remote Windows NT server. Stream files can also be stored remotely on a Local Area Network (LAN) server.

Directories and Current Directory

A *directory* is a special object that is used to locate objects by names specified by users. Each directory contains a list of objects that are attached to it, and that list may include other directories.

The *current directory* is the first directory in which the operating system locates files, and where it also stores temporary files and output files. When you request an operation for an object, such as a file, the system searches for the object in the current directory, unless a different directory path is specified. The current directory is similar in nature to the current library. If the file selection does not start with '/' (Root Directory), the files should be in the path of the current directory.

Path and Path Names

A *path name* (also called a *pathname* on some systems) informs the system how to locate an object. The path name is expressed as a sequence of directory names followed by the name of the object. Individual directories and the object name are separated by a slash (/) character. An example might be: directory1/directory2/file.

For convenience, the back slash (\) can be used instead of the slash in integrated file system commands.

There are two ways of indicating a path name:

An *absolute path name* begins at the highest level, or *root directory* (which is identified by the / character). For example, consider the following path from the / directory to the file named *testit*: */mydept/myfiles/testit*.

If the path name does not begin with the / character, the system assumes that the path begins at your current directory. This type of path name is called a *relative path name*. For example, if your current directory is *mydept* and it has a sub-directory named *myfiles* containing the file *testit*, the relative path name to the file is: *myfiles/testit.* Notice that the path name does not include the name of the current directory. The first item in the name is the directory or object at the next level below the current directory.

Stream Files

A *stream file* is a randomly accessible sequence of bytes with no further structure imposed by the system. The integrated file system provides support for storing and operating on information in the form of stream files. Documents that are stored in IBM i folders are stream files. Other examples of stream files are PC files and the files in UNIX systems. An integrated file system stream file is a system object that has an object type of *STMF.

Other IFS Objects

There are other object types (such as link objects, etc.) in the IFS which at this time are not supported by ***Smartcrypt^{i}.***

File Systems in the IFS

There are currently ten (10) file systems that are part of the Integrated File System. Each file system is a major sub-tree in the IFS directory structure. A file system provides the support to access specific segments of storage that are organized as logical units. These logical units on the IBM i are files, directories, libraries, and objects.

Each of these file systems has a set of logical structures and rules for interacting with information in storage. These structures and rules may be (and often are) different from one file system to another. The IFS treats the library support and folders support as separate file systems.

The ten file systems are:

- **"root"** - / **file system.** This file system takes full advantage of stream file support and hierarchical directory structure of the integrated file system. The root file system has the characteristics of the Disk Operating System (DOS) and OS/2 file systems. Most of references throughout this guide refer to the "root" system.
- **QDLS** - **Document Library Services file system.** This file system provides access to documents and folders. See IBM's *Office Services Concepts and Programmer's Guide (SH21-0703)* for additional information.
- **QOPT** - **Optical file system.** This file system provides access to stream data that is stored on optical media (such as CDs). See IBM's *Optical Support (SC41-5310)* for additional information.
- **QSYS.LIB** - **Library file system.** This file system supports the IBM i library structure and provides access to database files and all of the other IBM i object types that the library support manages.

- **NFS** - **Network File System.** This file system provides the user with access to data and objects that are stored on a remote NFS server. An NFS server can export a network file system that NFS clients will then mount dynamically. See IBM *i OS Network File System Support (SC41-5714)* for additional information.
- **QFileSvr.400.** This file system provides access to other file systems that reside on remote IBM i systems. See IBM's *Integrated File System Introduction (SC41-5711)* for additional information.
- **QNetWare** - **QNetWare file system.** This file system provides access to local or remote data and objects that are stored on a server that runs Novell NetWare 4.10 or 4.11 or to standalone PC servers running Novell Netware 3.12, 4.10, 4.11, or 5.0. A user can mount NetWare file systems over existing local file systems dynamically. See *File Management (SC41-5710)* for additional information.
- **QNTC Windows NT Server file system.** This file system provides access to data and objects that are stored on a server running Windows NT 4.0 or higher. It allows IBM i applications to use the same data as Windows NT clients. This includes access to the data on a Windows NT Server that is running on an integrated PC Server. See IBM's *OS/400-iSeries Integration with Windows NT Server (SC41-5439)* for details.
- **QOpenSys** - **Open Systems file system.** This file system is compatible with UNIX-based open system standards, such as POSIX and XPG. Like the root file system, this file system takes advantage of the stream file and directory support that is provided by the integrated file system. In addition, it supports case-sensitive object names. See IBM's *Integrated File System Introduction (SC41-5711)* for additional information.
- **UDFS** - **User-Defined File System.** This file system resides on the Auxiliary storage pool (ASP) of the user's choice. The user creates and manages this file system. See IBM's *Integrated File System Introduction (SC41-5711)* for additional information.

***Smartcrypt**^{i}* works with all file systems, but the rules of each file system must be adhered to or a file I/O error will most likely occur. In most cases, the files can be compressed and extracted in one run when all the file names and paths meet the file system's rules. When creating an archive file in one file system, one restriction is that when using the TMPPATH option, the temp path must also be in the same file system as the archive files.

On the following pages are rules for some of the most used file systems.

Document Library Services File System (QDLS)

The QDLS file system supports the folders structure. It provides access to documents and folders. Additionally, it supports IBM i folders and document library objects (DLOs) and supports data stored in stream files.

Considerations and Limitations:

- You must be enrolled in the system distribution directory when working with objects in QDLS.
- QDLS converts the lowercase English alphabetic characters *a* through *z* to uppercase when used in object names. Therefore, a search for object

names using only those characters is not case sensitive. All other characters are case sensitive in QDLS.

- Each component of the path name can consist of just a name, such as: */QDLS/MYFLR1/MYDOC1* - or - a name plus an extension (similar to a DOS file extension), such as: */QDLS/MYFLR1/MYDOC1.TXT*.
- The name in each component can be up to 8 characters long, and the extension (if any) can be up to 3 characters long. The maximum length of the path name is 82 characters, assuming an absolute path name that begins with */QDLS*.
- The directory hierarchy within QDLS can be 32 levels deep.
- Must have proper authority within the path.
- The folders in the path must already exist.
- PKZIP will not create folders at this time.
- For more details, see the "Rules for Specifying Folder and Document Names" discussion in the publication *CL Reference*.

Creating an Archive in a QDLS Personal Folder

The following is an example of creating and processing an archive in the Document Library Services file system. First, assume a folder in QDLS with a name of MYFOLDER where the archives will be stored. To view the folders, issue the command WRKLNK '/QDLS/*' (you could use WRKDOC and WRKFLR, but WRKLNK is better to use since PKZIP will be using /QDLS).

```
                             Work with Object Links

Directory  . . . . :   /qdls

Type options, press Enter.
  3=Copy   4=Remove   5=Next level   7=Rename   8=Display attribu
  11=Change current directory ...

Opt   Object link            Type             Attribute    Text
      .                      FLR
      ..                     FLR
      MYFOLDER               FLR
      QBKBOOKS               FLR
```

Run the PKZIP command:

PKZIP ARCHIVE('/QDLS/MYFOLDER/MYARCH1.ZIP') FILES('testlib/ben') TYPARCHFL(*IFS)

The suffix .ZIP was added to help identify the file as an archive file.

```
Scanning files for match  ...
Found  1 matching files
Compressing TESTLIB/BEN(BEESON) in TEXT mode
Add  TESTLIB/BEN/BEESON  --   Deflating (32%)
PKZIP Compressed 1 files in Archive /QDLS/MYFOLDER/MYARCH1.ZIP
PKZIP Completed Successfully
Press ENTER to end terminal session.
```

To see the file in the folders, run WRKLNK '/QDLS/MYFOLDER/*'

```
                             Work with Object Links

 Directory  . . . . :   /QDLS/MYFOLDER

 Type options, press Enter.
   3=Copy   4=Remove   5=Next level   7=Rename   8=Display attributes
   11=Change current directory ...

 Opt   Object link            Type             Attribute    Text
       .                      FLR
       ..                     FLR
       MYARCH1.ZIP            DOC
```

Next, to view the contents, run:

PKUNZIP ARCHIVE('/QDLS/MYFOLDER/MYARCH1.ZIP') TYPARCHFL(*IFS)

```
Archive:  /QDLS/MYFOLDER/MYARCH1.ZIP   551 bytes   1 file
  Length  Method    Size  Ratio  Date    Time  CRC-32    Name
 -------- ------  ------- -----  ----    ----  ------    ----
      259 Defl:F      177  32% 11-27-00 15:32 b5dbf80c TESTLIB/BEN/BEESON
 --------         -------  ----                         -------
     259              177  32%                          1 file
PKUNZIP extracted     0 files
PKUNZIP Completed Successfully
Press ENTER to end terminal session.
```

Optical File System (QOPT)

The QOPT file system provides access to stream data that is stored on optical media (such as CDs). Additionally, it provides a hierarchical directory structure (similar to PC operating systems such as DOS and OS/2), is optimized for stream file input/output, and supports data stored in stream files (known as DSTMF or Distributed Stream Files).

Considerations and Limitations:

- QOPT converts the lowercase English alphabetic characters a to z to uppercase when used in object names. Therefore, a search for object names using only those characters is not case-sensitive. For more details, see the publication *Optical Support (SC41-5310)*.
- The path name must begin with a slash (/) and contain no more than 294 characters. The path is made up of the file system name, the volume name, the directory and sub-directory names, and the file name. For example: /QOPT/VOLUMENAME/DIRECTORYNAME/SUBDIRECTORYNAME/FILENAME
- The file system name (/QOPT) is required.
- The volume name is required and can be up to 32 characters long.
- You can include one or more directories or sub-directories in the path name, but QOPT requires none. The total number of characters in all directory names and sub-directory names (including the leading slash) cannot exceed 256 characters. Directory and file names allow any character except X'00' through X'3F', X'FF', lowercase alphabetic characters, and the following characters:
- Asterisk (*)

- Hyphen (-)
- Question mark (?)
- Quotation mark (")
- Greater than (>)
- Less than (<)
- The file name is the last element in the path name. The file name length is limited by the directory name length in the path. The directory names and file name combined cannot exceed 256 characters, including the leading slash.

Processing Archive on a CD (QOPT)

The following is an example of processing an archive that exists on a CD and using PKUNZIP to view or extract. Because the archive file is on a CD, and the file system QOPT controls the CD, this archive basically exists in the IFS.

First, check and ensure the archive is on the CD by doing a WRKLNK (you can use WRKOPTDIR, but using WRKLNK will show the actual paths required). Remember, the volume of the CD is also a directory in QOPT file system. If the file names are longer than eight characters, the file name will be changed, much like you see in DOS systems. It will contain a tilde (~) followed by a number for files found with excessive name lengths.

WRKLNK '/QOPT/*'

```
                              Work with Object Links

 Directory  . . . . :   /QOPT

 Type options, press Enter.
   3=Copy   4=Remove   5=Next level   7=Rename   8=Display attributes
   11=Change current directory ...

 Opt   Object link            Type             Attribute    Text
       MYTESTLABEL            DDIR
```

The above screen shows that the volume label of the CD is "MYTESTLABEL". Using the "5" for the next level option, you can navigate through the directories. You will then see the files and directories on the root of the CD. For example:

```
                              Work with Object Links

 Directory  . . . . :   /QOPT/MYTESTLABEL

 Type options, press Enter.
   3=Copy   4=Remove   5=Next level   7=Rename   8=Display attributes
   11=Change current directory ...

 Opt   Object link            Type             Attribute    Text
       ARCHIVE.ZIP            DSTMF
       GZIPPW.GAR             DSTMF
       OS_400~3.DOC           DSTMF
       PKZCVT~2.DOC           DSTMF
       PKW90~1.SAV            DSTMF
       PKW90~1.ZIP            DSTMF
```

To view the archive PKW90~1.ZIP (which is really the long name PKW14053S.ZIP) contents, use PKUNZIP with *VIEW.

Use the command:

PKUNZIP ARCHIVE('/QOPT/MYTESTLABEL/PKW90~1.ZIP') TYPARCHFL(*IFS) TYPE(*VIEW)

```
Archive:  /QOPT/MYTESTLABEL/PKW90~1.ZIP   1373026 bytes   1 file
  Length  Method    Size  Ratio  Date    Time  CRC-32    Name
 -------- ------  ------- -----  ----    ----  ------    ----
  6044544 Defl:N  1372902  77% 08-16-01 21:17 d73f09cf PKW14053S.sav
 --------         -------  ----                         -------
 6044544          1372902  77%                          1 file
PKUNZIP extracted     0 files
PKUNZIP Completed Successfully
```

Compressing files from a CD (QOPT)

We can compress the document (.DOC) files on the CD shown in the previous example and store them in an archive. Use this command to store in my archive library ATEST under the file V509 archives with an archive file member named CDTEST01.

PKZIP ARCHIVE('atest/v509/cdtest01') FILES('/QOPT/MYTESTLABEL/OS_400~3.DOC' '/QOPT/MYTESTLABEL/PKZCVT~2.DOC') TYPFL2ZP(*IFS)

```
Scanning files for match  ...
Found  2 matching files
Compressing /QOPT/MYTESTLABEL/OS_400~3.DOC in BINARY mode
Add  /QOPT/MYTESTLABEL/OS_400~3.DOC  --   Deflating (77%)
Compressing /QOPT/MYTESTLABEL/PKZCVT~2.DOC in BINARY mode
Add  /QOPT/MYTESTLABEL/PKZCVT~2.DOC  --   Deflating (79%)
PKZIP Compressed 2 files in Archive ATEST/V509(CDTEST01)
PKZIP Completed Successfully
```

Because you would not be able to extract them to the CD, you may want to use the parameter STOREPATH(*NO) so that only file names OS_400~3.DOC and PKZCVT~2.DOC are stored in the archive.

For more details on path name rules in the QOPT file system, see the “Path Name Rules” discussion in the publication *Optical Support (SC41-5310)*.

Using QSYS.LIB via the Integrated File System Interface

Even though ***Smartcrypt**^i^* accesses the QSYS library file system directly, there is an ability to access the QSYS.LIB file system through the Integrated File System interface. In using the Integrated File System interface, be aware of the following considerations and limitations:

- Logical files are not supported.
- Physical files supported for text mode access are program-described physical files containing a single field and source physical files containing a single text field. Physical files supported for binary mode access include externally-described physical files in addition to files supported for text mode access.

- If any job has a database file member open, only one job is given write access to that file member at any given time. Other requests are allowed read-only access.
- In general, the QSYS.LIB file system does not distinguish between uppercase and lowercase in the names of objects. A search for object names achieves the same result, regardless of whether characters in the names are uppercase or lowercase. If a name is enclosed in quotation marks, the case of each character in the name is preserved. A search involving quoted names, therefore, is sensitive to the case of the characters in the quoted name.
- Each component of the path name must contain the object name followed by the object type of the object. For example: */QSYS.LIB/TESTLIB.LIB/MYFILE.FILE/MYFILE.MBR*. The object name and object type are separated by a period (.). Objects in a library can have the same name if they are different object types, so the object type must be specified uniquely to identify the object.
- The object name in each component can be up to 10 characters long, and the object type can be up to 6 characters long.
- The directory hierarchy within QSYS.LIB can either be two or three levels deep (two or three components in the path name), depending on the type of object being accessed. If the object is a database file, the hierarchy can contain three levels (library, file, or member), otherwise, there can be only two levels (library or object). The combined length of each component name plus the number of directory levels determines the maximum length of the path name. If / and QSYS.LIB are included as the first two levels, the directory hierarchy for QSYS.LIB can be up to five levels deep.
- The characters in names are converted to code when the names are stored. Quoted names, however, are stored using the code page of the job.

For information about code pages, see the publication *National Language Support*.

IFS Summary

Only directories and stream files are supported by ***Smartcrypt^{i}.***

If the archive file is to be in IFS, set parameter TYPARCHFL(*IFS).

If the file being compressed or extracted is in IFS, set parameter TYPFL2ZP(*IFS)

If the files to be selected for compression are to be non-case sensitive set parameter TYPARCHFL(*IFS2).

If the list files are to be in IFS (see Appendix C), set parameter TYPLISTFL(*IFS).

Format Summary:

Directory	Directory1/directory2	will be current directory
Stream File	filename or directory/filename	will be current directory
Full Path	/Directory1/Directory2/filename	

For more information, see the IBM publication *Integrated File System Introduction* (SC41-5711) or visit the IBM web site.

SAVF

SAVF, denoted by the IBM i OS system TYPE(*FILE) and ATTR(SAVF), is a special form of file designed specifically to handle save/restore data in the IBM i system.

Some SAVF special characteristics are:

- The SAVF is always processed as binary with all records being 528 characters in length.
- Only a save and restore IBM i function can update or change data.
- A SAVF will <u>not</u> be selected if a member name is included in the file specification.
- A SAVF is a means to compress other IBM i object types (programs, modules, commands, logical files, triggers, etc.) that are in the IBM i system by first doing a SAVLIB or SAVOBJ for those objects to a SAVF. Then you can compress and extract the SAVF.

Compressing a SAVF file

The only difference when compressing a SAVF is not to specify a member (only library/file). If a member is specified, then no SAVF types will be compressed.

Extracting Records into a SAVF file

It is helpful before extracting records from a ZIP archive to be aware of what file names and file attributes are being stored for the compressed file. VIEWOPT(*DETAIL) may be used on the archive to verify the information. An attribute is stored in the archive header that identifies if the file is a SAVF. The PKUNZIP program will also retain the original attribute from the extended attributes, such as SAVF description and library description.

A common problem in some IBM i environments is that some users may not have the authority to the SAVF commands which can result in failures.

Overwriting Current SAVF File

When extracting a compressed file, it may be desirable to overwrite the existing file. By using the OVERWRITE(*YES) parameter, PKUNZIP will first issue a CLRSAVF command to clear the save file. This demonstrates why care should be taken when extracting a SAVF.

Compressing Spool Files

Smartcrypt^{i} has the ability to select, compress and extract spool files. Not only can a spool file be compressed, they can be converted to other document formats that will allow the document file to be distributed and read by other media and software.

All spool files are eligible for compression but only spool file types *SCS, *IPDS are supported for text document conversion.

By using the PKZIP command and setting parameter TYPFL2ZP(*SPL), other parameters will be shown to help select the spool files. To assist, a new command PKZSPOOL is provided to sequence the selections and to eliminate parameters that are not valid for the selection of spool files.

Spool file parameters specify the group of spool files that are to be selected. Eight positional values can be specified to select the spool files: the spool file name (SPLFILE), the spool file number (SPLNBR), the user that created the files (SFUSER), the OUTQ that the file is residing (SFQUEUE), the form type specified (SFFORM), the user data tag associated with the spool file (SFUSRDTA), the status of the spool file (SFSTATUS), or the specific job name/user name/job number (SFJOBNAM). Only files that meet all of the selection values will be selected. A sample of the default selection parameters is shown in the window below:

Selection sample using the PKZSPOOL command.

```
                    SPLF File Compression   (PKZSPOOL)

 Type choices, press Enter.

 Archive Zip File name  . . . . . > myar



 Spool File . . . . . . . . . . .   *ALL          Spool File Name,  *All
 Spool File User  . . . . . . . .   *CURRENT      User ID, *CURRENT, *ALL
               + for more values
 Output Queue Name  . . . . . . .   *ALL          OutQ name, *ALL
   Library  . . . . . . . . . . .   _________          Library, *LIBL, *CURLIB
 Print Form Type  . . . . . . . .   *ALL          Form Type, *STD, *ALL
 Print File User Specified Data     *ALL          User Data,  *all
 Spool Files Status . . . . . . .   *ALL          *ALL, *READY, *HELD...
               + for more values
 Spool File Job Name  . . . . . .   _________     Job name, blank for all
   User . . . . . . . . . . . . .   _________     User Id
   Job Number . . . . . . . . . .   _______       Job Number
 Spooled file number  . . . . . .   *ALL          1-9999, *ALL, *LAST
 Target File Format . . . . . . .   *SPLF         *SPLF, *TEXT, *PDF, *TEXT1...
 Target File Name . . . . . . . .   *GEN1
 Type of processing . . . . . . .   *ADD          *ADD, *UPDATE, *FRESHEN ..
 Compression Level  . . . . . . .   *SUPERFAST    *NO, *FAST, *NORMAL, *MAX...
 File Types . . . . . . . . . . .   *DETECT       *DETECT *TEXT *BINARY ........
 Zip Spool Files  . . . . . . . .   *SPL          *SPL
 Archive Passphrase . . . . . . . .
```

After defining what spool files are to be selected for compression, you will need to define the file format the spool file should be stored in the archive. At this time, there three formats: *SPLF (spool file native mode), *TEXT (ASCII text document with three variations of how a new page is handled) and *PDF (Adobe portable document format).

For use with *TEXT and *PDF there are three variations of storing the file name in the archive with the parameter SFTGFILE. SFTGFILE (*GEN1) will generate a very specific name using most of the spool file name attributes to form the file name so that it will not be a duplicated. The name will be built as follows:

> "Job-Name**/**User-Name**/** **#**Job-Number**/**Spool-File-Name**/** **F**spool- File-Number.Suffix"

For example: "*MYJOB/BILLS/#152681/INVOICE/F0021.SPLF*"

The suffix is dependent on the SFTARGET setting. *SPLF can only be stored as SFTGFILE (*GEN1).

SFTGFILE (*GEN1P) will generate the same specific name generated by *GEN1 except the '/' for folders will all be replaced by '.' to make the file name one lone name. For example:

MYJOB.BILLS.#152681.INVOICE.F0021.SPLF

SFTGFILE (*GEN2) uses the spool file name and appends the spool file number followed by the suffix that is depended on the SFTARGET setting. Caution should be taken in that a duplicate file name in the archive could be created. An example of GEN2 is a spool file INVOICE with spool file number of 21 that will be converted to a text file will generate a file name of INVOICE21.TXT.

In cases where a very specific name is desired for the file in archive name, SFTGFILE() can be coded with the name. This is designed for selecting only one file at a time otherwise file names will be duplicated. Alternatively, you could add coding to the CVTNAME routine and use the CVTFLAG to generate the desired file name.

Compressing Spool Files Examples

The following are several examples demonstrating the selection of spool files for compression.

Example 1: Select a specific spool file (MYSPLFFILE) for the specific job (jobname-WSSSPL, User-WSS and job number 11) in all output queues (the default of *SFQUEUE*) and convert the spool file to a PDF format *SFTARGET(*PDFLETTER)* to fit a letter format. store the archive in the IFS with *TYPARCHFL(*IFS)* .

```
PKZSPOOL ARCHIVE('/yourpath/bills/splftest01.zip') TYPARCHFL(*IFS)
         SPLFILE(MYSPLFILE) SFUSER(*ALL) SFJOBNAM(11/WSS/WSSSPL)
         SFTARGET(*PDFLETTER) SFTGFILE(*GEN1P)
```

Example 2: Select all spool files belonging to users WSS and TAIT (*SPLUSERID*) that resides in the OUTQ QPRINTS (*SFQUEUE*) and compress them as spool files with *SFTARGET(*SPLF)*. This might be done to save the spool files for later review since this OUTQ is purged on a regular basis.

```
PKZSPOOL ARCHIVE('/yourpath/bills/splftest02.zip') TYPARCHFL(*IFS)
         SFUSER(WSS TAIT) SFQUEUE(QPRINTS) SFTARGET(*SPLF) SFTGFILE(*GEN1)
```

Example 3: Using the archive from Example 2, we want to restore or extract the spool files in order to **print** them again. Except in this case we want them to belong to the user MAS with *SPLUSERID* and place the spool files in the OUTQ MASQ (*SFQUEUE*) located in the library DEVPLIB.

```
PKUNZIP ARCHIVE('/yourpath/bills/splftest02.zip') TYPARCHFL(*IFS)
        TYPE(*EXTRACT) SPLUSRID(MAS) SFQUEUE(DEVPLIB/MASQ)
```

Example 4: Select the spool file QPRINTS (*SPLFILE*), spool file number 17 (*SPLNBR*), user MAS (*SFUSER*) and convert the file to a TEXT file with *SFTARGET(*TEXTFC).* In this case, the file is needed to read into a PC program and the user wants the ANSI control characters in position 1 of each line.

```
 PKZSPOOL ARCHIVE('/yourpath/bills/splftest04.zip')  TYPARCHFL(*IFS)
     SPLFILE(QPRINTS) SFUSER(MAS) SPLNBR(17)
     SFTARGET(*TEXTFC) SFTGFILE(*GEN1P)
```

Example 5: Now we want to extract the text file created in Example 4 to one of our shared drives areas ('/yourpath/PCFILES') that our PCs can access. In this case the normal extraction would identify the file as a text file and would convert it to EBCDIC. Since the file will be used by a PC program that is expecting the data to be in ASCII, we will have to extract the file as binary since the internal file is already in ASCII. By specifying FILETYPE(*BINARY), this ensures that no translation of the data takes place.

```
PKUNZIP ARCHIVE('/yourpath/bills/splftest04.zip') TYPARCHFL(*IFS) TYPFL2ZP(*IFS)
TYPE(*EXTRACT) FILETYPE(*BINARY)
EXDIR('/yourpath/PCFILES') DROPPATH(*ALL)
```

6 IBM i PKWARE Save/Restore Application Feature (iPSRA)

The IBM i PKWARE Save/Restore Application (iPSRA) feature enables ***Smartcrypt*** ^{i} to compress/encrypt IBM i save files directly to a file in an archive. The process produces a result similar to creating a save file first and then compressing and/or encrypting it into an archive, but the iPSRA feature economizes on time and disk space by skipping the intermediate step. The iPSRA process can be integrated with your existing backup/recovery procedures and systems on the IBM i.

iPSRA assists not only with compressing your save data but with encrypting the data for offsite storage. The iPSRA process can execute multiple save operations with one compression run, making it unnecessary to run repeated individual save commands.

The iPSRA feature integrates ***Smartcrypt*** ^{i} compression and encryption technology with IBM i Save and Restore APIs. The iPSRA feature works with the PKZIP command to save objects and with the PKUNZIP command to restore them.

To use the iPSRA feature, you must have a working knowledge of the save/restore commands in their native mode. The same uses and restrictions apply to the save/restore commands in ***Smartcrypt*** ^{i} as to the native commands. The use and format of outfiles with any of the save or restore commands are the same as with the native commands. For information about the native save and restore commands, see the IBM manuals that describe these commands, or the IBM Web site:

http://publib.boulder.ibm.com/eserver/ibmi.html

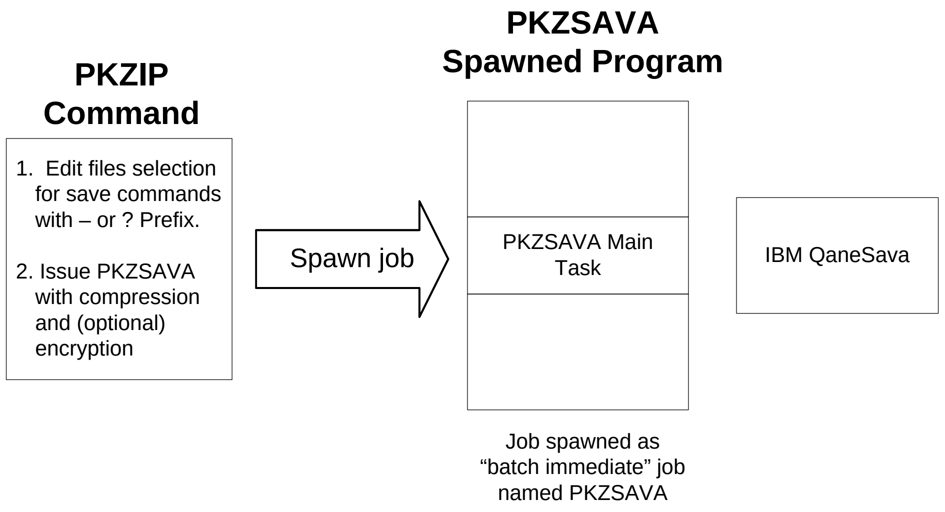
Save/Restore Command Overview

The iPSRA feature supports the following Save command types:

- Save (SAV) command
- Save Object (SAVOBJ) command
- Save Document Library Object (SAVDLO) command
- Save Library (SAVLIB) command
- Save Changed Object (SAVCHGOBJ) command

With the iPSRA feature, PKZIP spawns a batch immediate program named PKZSAVA that processes save command data and causes it to be compressed—and optionally encrypted—into an archive instead of being saved directly to disk. PKZSAVA uses the

IBM API, which is a pre-started job in the QSYSWRK subsystem. The figure below gives an overview of the process.


PKZIP
Command
1. Edit files selection for save commands with – or ? Prefix.
2. Issue PKZSAVA with compression and (optional) encryption
Spawn job
PKZSAVA
Spawned Program
PKZSAVA Main Task
IBM QaneSava
Job spawned as "batch immediate" job named PKZSAVA

Saving Data

To use the iPSRA feature to save data, you enter command strings for SAV, SAVOBJ, and/or SAVLIB in the FILES parameter of PKZIP. Prefix each command string with a hyphen (-) or question mark (?).

- A '-' tells PKZIP that the entire command is entered and is ready to be validated
- A '?' causes PKZIP to prompt for command parameters to enter before validating command syntax

The following example shows a PKZIP command to save the library MYLIB and some other objects:

➔ **PKZIP ARCHIVE('/MYpath/myarchive') FILES('-SAVLIB LIB(MYLIB) DEV(MYNAME1)' '?SAVOBJ DEV(MYNAME2)')**

The command line above creates a saved library file or iPSRA file of the library MYLIB and compresses it into the archive */MYpath/myarchive* as file SAVLIB01_MYNAME1. The command line uses the '?' prefix with the command SAVOBJ to prompt for the objects to be saved. These must be specified at the prompt. The SAVOBJ command creates an iPSRA file named SAVOBJ02_MYNAME2 in the archive.

Restoring Data

The restore of a save object with a RSTLIB command works similarly except that RSTLIB is used with PKUNZIP and only one file can be restored at a time, to assure proper building of the saved objects.

Examples are given at the end of this chapter.

Syntax

Save command parameters must be consistent with the save command type entered and must be separated by at least one blank character. Refer to the Control

Language (CL) documentation for detailed information about valid parameters when you save objects to save files.

File Names Used for Saved Data

The file name PKZIP uses for saved data in an archive is based on the type of save command and the name used in the save command DEV parameter. If the DEV parameter is *SAVF, then the name comes from the SAVF parameter. See the table below for examples.

Command	***File Name***
SAVOBJ DEV('MYPAYROLL')	SAVOBJnn_ MYPAYROLL
SAVLIB DEV(*SAVF)	SAVF(MYLIB/MYSAVF)
SAV DEV('MYSAVFDEVPATH')	SAVnn_ MYSAVFDEVPATH

The *nn* in the example file names is the sequence number of the command for one PKZIP run.

PKZIP removes from an archive existing files that have the same name as a new save file. To help you avoid duplicating file names when updating an existing archive, PKZIP checks to see if there are any iPSRA files in the archive. If there are, PKZIP uses the largest *nn* number plus 1 as the starting *nn* for iPSRA files to be added. This helps avoid accidentally overwriting iPSRA files when updating an archive.

Extended Data in Archive

Each save operation creates specific data in the extended data area to tell PKUNZIP that a file to be restored is a file of the special SAVE type. The extended data also provides history information that can be displayed with VIEWOPT(*ALL) to show the original command, the job when the save data was created, the target release used for the save, and the spawned job PKZSAVA.

```
Filename: SAVLIB01_DEDSAV01
Detected File type:     SAVE Apps. Data
Created by:             PKZIP(R) for IBM i  14.0
Zip Spec to Extract:    2.0  Or Greater
Compression method:     Deflated   [Superfast]
Date and Time           2009 Aug 8 14:08:58
Compressed size:        42876 bytes
Uncompressed size:      413808 bytes
32-bit CRC value (hex): 334d1674
Extended attributes:    yes, [Length = 134]
File Save Apps Data:<savlib lib(deD) dev(dedsav01) OUTPUT(*OUTFILE)
  OUTFILE(WSS/TEM01)>
                   :TGTRLS(*Current) Save Job <019627/USER1/PKZSAVA>
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 1 file, 413808 bytes uncompressed, 42876 bytes compressed:   90%
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

```
                          Additional Message Information

 Message ID . . . . . . :   AQZ0895
 Date sent  . . . . . . :   08/08/09      Time sent  . . . . . . :   14:09:20

 Message . . . . :   File Save Apps Data:<savlib lib(deD) dev(dedsav01)
```

```
  OUTPUT(*OUTFILE) OUTFILE(WSS/TEM01)>

This shows that the file is a Save Application data file type 4 and the
  command that was used to store the save file in the archive.  Save Data
  created with PKZIP job 019586/USER1/USER1L01. Target System was *Current.
```

Notice in the output above that the spawned job (019627/USER1/PKZSAVA) was captured as well as the PKZIP job (019586/USER1/USER1L01). Information on the spawned job may be needed to do a DSPLOG command.

The target release (TGTRLS) is also shown to note the target setting for the run.

Notes and Restrictions

All IBM restrictions and security requirements apply to the use of the Save and Restore commands in iPSRA. Some additional restrictions are noted below.

PKZIP

- QTEMP cannot be specified for the library name on the OUTFILE or SAVACTMSGQ parameters.
- Some parameters of the save commands that are not used by iPSRA are ignored. For example, CLEAR, DTACPK, and so on.
- Objects saved by PKZIP can only be restored using the restore from application with PKUNZIP, and they can only be restored to a release of the operating system that is the same or later than the version from which the objects were saved.
- The save parameters are only completely validated when PKZIP submits the save command for processing.
- A target release VxRxMx value prior to V5R1M0 is not valid: The iPSRA feature is not supported prior to version 5, release 1, modification level 0 of PKZIP. The version, release, and modification level depend on the save operation being performed. See the valid values for the TGTRLS parameter table in the iSeries Backup and Recovery Manual for a complete list of valid values.
- All compression methods except the Terse compression method are supported with iPSRA.
- Positional options on the save commands must contain the parameter. For example, the save library command

  SAVLIB WSS DEV(*SAVF) SAVF(TESTWSS)

  would not work to save the library WSS. The WSS is a positional parameter, where it is assumed WSS was the LIB option. The correct approach with iPSRA is to use the command:

  -SAVLIB LIB(WSS) DEV(*SAVF) SAVF(TESTWSS)
- The save operation must be completely successful or it aborts. If any object is not saved for any reason, the PKZIP job assumes a failure. Do not include an object that will not save, as this object will cause a major failure.

PKUNZIP

- QTEMP cannot be specified for the library name on the OUTFILE parameter.
- To ensure that all objects are properly restored, only one restore command can be processed per run.
- Some parameters of the restore commands are not used by iPSRA and are ignored. For example, VOL, SEQNBR, and so on.
- The user must have the required security for the restore command.

Using OUTPUT and OUTFILE with the Save Commands

A save command can have an OUTFILE parameter that is used to build a file listing the objects saved with that command. When an OUTFILE is specified for a save command, PKZIP automatically archives the outfile in the same archive as the iPSRA file with the name specified. The outfile provides a record of the contents of an iPSRA file. An outfile has the format and restrictions defined by IBM for the save commands. Use of outfiles is optional.

If OUTPUT(*PRINT) or OUTFILE(*PRINT) is used with a save or restore command, the printout is produced by the IBM API job – not with the PKZIP job or the spawned PKZSAVA job. Therefore, it appears in the special OUTQ job named QPRTJOB for each user.

How to Use the Save Application Feature

The save option is activated by entering a save command in the FILES parameter of PKZIP, prefixed with a hyphen (-) or question mark (?). Multiple save commands can be entered to select multiple sets of files in the same pass. The save commands do not need to be the same, nor do they need to use the same prefix.

If a command fails the pre-command processor, PKZIP issues the message AQZ0332, which shows the failed command. The reason for the failure appears in the job log prior to this message.

If a failure occurs during the processing of the save commands, the reason for the failure appears in the job log of the spawned job. If any errors occur in the spawned job, a job log will be forced. There is no pre-check processing on security or on the objects themselves. All data verification is handled by the save API.

For example, the following command tries to save a library (NOLIB) that does not exist:

➔ **PKZIP ARCHIVE('/yourpath/BILLS/X5TESTL.ZIP') TYPARCHFL(*IFS) FILES('-SAVLIB LIB(NOLIB) DEV(TESTSAV01) OUTPUT(*PRINT) ')**

The log of the PKZIP output might look like this:

```
Scanning files in *DB for match  ...
Found  0 matching files
1 Save Command(s) selected
Command:<SAVLIB LIB(NOLIB) OUTPUT(*PRINT)>
Compressing SAVLIB01_TESTSAV01 in SAVE Apps. Data mode
```

```
Save Operation encountered an error.  See Job Log of PKZSAVA save job for fur
ther details.
iPSRA Initialization Failure has occurred
iPSRA Failed.  Save command not successful.
Smartcrypt Copied 1 files from input archive
Smartcrypt Compressed 0 files in Archive /yourpath/BILLS/X5TESTL.ZIP
Smartcrypt Completed with Errors
Press ENTER to end terminal session.
```

The job log of the PKSAVF output might look like this:

```
CPF3781    Diagnostic 30   08/08/09  14:49:10.428528  QANESERV     QSYS
From module . . . . . . . . :   QANESERV
From procedure  . . . . . . :   QaneSendPgmMsg__FP14qanec_CTLBLK_tPcT2iN24
Statement . . . . . . . . . :   19
To module . . . . . . . . . :   QP0ZPCPN
To procedure  . . . . . . . :   InvokeTargetPgm__FP11qp0z_pcp_cb
Statement . . . . . . . . . :   187
Message . . . . :   Library NOLIB not found.
   Cause . . . . . :   The library specified for the save or restore
   operation does not exist on the system. Recovery  . . . :   Do one of the
   following and try the request again: If this is a save operation, correct
   the library name on the LIB parameter. If this is a restore operation,
   correct the library name specified on the SAVLIB or RSTLIB parameter, or
   use the Create Library (CRTLIB) command to create the library by
   specifying LIB(NOLIB). If this is a restore operation and VOL(*SAVVOL)
   was specified, the save library must exist in the auxiliary storage pool
   specified on the RSTASPDEV parameter. If RSTASPDEV(*SAVASPDEV) and
   RSTASP(*SAVASP) are specified along with VOL(*SAVVOL), then the save
   library must exist in the system ASP. To restore a library that is new to
   the system, specify VOL(*MOUNTED) instead of VOL(*SAVVOL).
```

How to Use the Restore Application Feature

To restore an iPSRA file from archive, code the restore command in the RSTIPSRA parameter of PKUNZIP. The RSTIPSRA parameter is defined as a command entry, so do not use quotes around the command. Enclose the entire restore command in parentheses: RSTIPSRA(command). To be prompted at the command, place the cursor on the RSTIPSRA entry and press the F4 key.

If an archive contains more than one file, you must use the FILES parameter to select the file you want to match up with the RSTIPSRA parameter. PKUNZIP restores only one iPSRA file per run.

If any object is not restored, PKUNZIP issues the message AQZ1007 and creates a job log for the PKZRSTA job. You should review the log to find any object that was not restored and the reason.

If a partial restore is performed, then the CRC and/or hash calculation for authentication does not take place, and the warning message AQZ1000 is displayed. This situation can arise if the save operation was a SAVLIB, but the restore operation restores only a few objects with the RSTOBJ.

Database Considerations for Save and Restore

Below are some tips for working with the save and restore functions.

- When you save an object to a save file or using iPSRA, you can prevent the system from updating the date and time of the save operation by specifying UPDHST(*NO) on the save command.
- When you restore an object, the system always updates the object description with the date and time of the restore operation. Display the object description and other save/restore related information by using the Display Object Description (DSPOBJD) command with DETAIL(*FULL).
- To display the last save/restore date for a database file, type: DSPFD FILE(filename) TYPE(*MBR).

Sample Jobs

iPSRA Example 1

The following example saves the library DED and prints the output of the save. It also saves the file object TESTFILE from the library TESTLIB with several options of the SAVOBJ. These save application files are compressed with a default setting and will be encrypted using a passphrase.

➔ **PKZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS) FILES(**
'-SAVLIB LIB(DED) DEV(DEDSAV01) OUTPUT(*PRINT) '
'-SAVOBJ OBJ(TESTFILE) LIB(TESTLIB) DEV(TESTOBJ11) OBJTYPE(*FILE) TGTRLS(V5R1M0) UPDHST(*NO) PRECHK(*YES) OUTPUT(*PRINT) ')
PASSWORD('bills00000') VPASSWORD('bills00000')

```
Scanning files in *DB for match  ...
Found  0 matching files
2 Save Command(s) selected
Command:<SAVLIB LIB(DED)  OUTPUT(*PRINT)>
Compressing SAVLIB01_DEDSAV01 in SAVE Apps. Data mode
Add  SAVLIB01_DEDSAV01  --  Deflating (90%)  encrypt(AES 256Key)
Command:<SAVOBJ OBJ(TESTFILE) LIB(TESTLIB) OBJTYPE(*FILE)
UPDHST(*NO) PRECHK(*YES) OUTPUT(*PRINT)>
Compressing SAVOBJ02_TESTOBJ11 in SAVE Apps. Data mode
Add  SAVOBJ02_TESTOBJ11  --  Deflating (79%)  encrypt(AES 256Key)
Smartcrypt Compressed 2 files in Archive /yourpath/bills/testsavx1.zip
Smartcrypt Completed Successfully
```

iPSRA Example 2

The following commands display the contents of the archive:

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS) TYPE(*VIEW)**

```
Archive:  /yourpath/bills/testsavx1.zip, 1358415 bytes, 2 files, 1 Segment
  Length Method    Size Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
   430192 Defl:S    43718  90% 08-09-05 08:16 ac1f8407 !SAVLIB01_DEDSAV01
  6325776 Defl:S  1313814  79% 08-09-05 08:16 101311d4 !SAVOBJ02_TESTOBJ11
 --------         -------  ----                          -------
 6755968          1357532  80%                           2 files
```

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS) TYPE(*VIEW) VIEWOPT(*ALL)**

```
Archive Comment:"Smartcrypt for IBM i"
Filename: SAVLIB01_DEDSAV01
Detected File type:     SAVE Apps. Data   Encrypt=Strong Encrypted
Created by:             PKZIP(R) for IBM i  14.0
Zip Spec to Extract:    5.1  Or Greater
Compression method:     Deflated   [Superfast]
Date and Time           2009 Aug 9 08:16:16
Compressed size:        43718 bytes
Uncompressed size:      430192 bytes
32-bit CRC value (hex): ac1f8407
Extended attributes:    yes, [Length = 130]
Strong Encryption AES 256 Key.
Algorithm Key 256,  Security type Passphrase
Number Certificate Recipients 0
Recipient List:
File Save Apps Data:<SAVLIB LIB(DED) DEV(DEDSAV01)  OUTPUT(*PRINT)>
                   :TGTRLS(*Current) Save Job <019700/USER1/PKZSAVA>
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Filename: SAVOBJ02_TESTOBJ11
Detected File type:     SAVE Apps. Data   Encrypt=Strong Encrypted
Created by:             PKZIP(R) for IBM i  9.2
Zip Spec to Extract:    5.1  Or Greater
Compression method:     Deflated   [Superfast]
Date and Time           2009 Aug 9 08:16:16
Compressed size:        1313814 bytes
Uncompressed size:      6325776 bytes
32-bit CRC value (hex): 101311d4
Extended attributes:    yes, [Length = 217]
Strong Encryption AES 256 Key.
Algorithm Key 256,  Security type Passphrase
Number Certificate Recipients 0
Recipient List:
File Save Apps Data:<SAVOBJ OBJ(TESTFILE) LIB(TESTLIB) DEV(TESTOBJ11) OBJTYPE
(*FILE) TGTRLS(V5R1M0) UPDHST(*NO) PRECHK(*YES) OUTPUT(*PRINT)>

                   :TGTRLS(*Current) Save Job <019701/USER1/PKZSAVA>
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 2 files, 6755968 bytes uncompressed, 1357532 bytes compressed:   80%
```

iPSRA Example 3

Now we want to restore the saved library DED to a new library called DEDNEW and then restore the object TESTFILE to the new DEDNEW library. PKUNZIP can perform only one restore at a time, so the operation requires two steps.

Step1.

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS) FILES('SAVLIB01_DEDSAV01') TYPE(*EXTRACT) RSTIPSRA(RSTLIB SAVLIB(DED) DEV(RSTDED) output(*print) RSTLIB(DEDNEW)) PASSWORD('bills00000')**

```
UNZIP Archive: /yourpath/bills/testsavx1.zip
Archive Comment:"Smartcrypt for IBM i"
Searching Archive /yourpath/bills/testsavx1.zip  for files to extract
Command:<RSTLIB SAVLIB(DED) RSTLIB(DEDNEW) OUTPUT(*PRINT)>
Extracting file SAVLIB01_DEDSAV01
```

```
Inflating *iPSRA:SAVLIB01_DEDSAV01 iPSRA File
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

Step 2.

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS) FILES('SAVOBJ02_TESTOBJ11') TYPE(*EXTRACT) RSTIPSRA(RSTOBJ OBJ(TESTFILE) SAVLIB(TESTLIB) DEV(RSTTEST) OBJTYPE(*FILE) RSTLIB(DEDNEW) OUTPUT(*PRINT)) PASSWORD('bills00000')**

```
UNZIP Archive: /yourpath/bills/testsavx1.zip
Searching Archive /yourpath/bills/testsavx1.zip  for files to extract
Command:<RSTOBJ OBJ(TESTFILE) SAVLIB(TESTLIB) OBJTYPE(*FILE)
  RSTLIB(DEDNEW) OUTPUT(*PRINT)>
Extracting file SAVOBJ02_TESTOBJ11
Inflating *iPSRA:SAVOBJ02_TESTOBJ11 iPSRA File
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

iPSRA Example 4

The following example shows that we can restore one or more objects from an iPSRA file that was created with SAVLIB.

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS) FILES('SAVLIB01_DEDSAV01') TYPE(*EXTRACT) RSTIPSRA(RSTOBJ OBJ(MYFILE2) SAVLIB(DED) DEV(RST1FILE) OBJTYPE(*FILE) RSTLIB(DEDNEW) OUTPUT(*PRINT)) PASSWORD('bills00000')**

```
UNZIP Archive: /yourpath/bills/testsavx1.zip
Searching Archive /yourpath/bills/testsavx1.zip  for files to extract
Command:<RSTOBJ OBJ(MYFILE2) SAVLIB(DED) OBJTYPE(*FILE)
RSTLIB(DEDNEW) OUTPUT(*PRINT)>
Extracting file SAVLIB01_DEDSAV01
Inflating *iPSRA:SAVLIB01_DEDSAV01 iPSRA File
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

iPSRA Example 5

The following example demonstrates the use of OUTFILE in a save command and shows how PKZIP automatically adds the outfile to the archive.

➔ **PKZIP ARCHIVE('/yourpath/bills/iPSRA_test/x3.zip')
TYPARCHFL(*IFS) FILES(
    '-SAVLIB LIB(DED) DEV(DEDSAV01) OUTPUT(*OUTFILE)
    OUTFILE(ATEST/DEDSAV01)'
    '-SAV DEV('IFS_testpkcs7*') OBJ(('/ajunk/testpkcs7/*'))
OUTPUT('/yourpath/bills/iPSRA_test/File01_SAV') ')**

```
Scanning files in *DB for match  ...
Found  2 matching files
2 Save Command(s) selected
```

```
Command:<SAVLIB LIB(DED) OUTPUT(*OUTFILE) OUTFILE(ATEST/DEDSAV01)>
Compressing SAVLIB01_DEDSAV01 in SAVE Apps. Data mode
Add  SAVLIB01_DEDSAV01  --  Deflating (90%)
Compressing ATEST/DEDSAV01(DEDSAV01) in TEXT mode
Add  ATEST/DEDSAV01/DEDSAV01  --  Deflating (98%)
Command:<SAV OBJ(('/AJUNK/TESTPKCS7/*'))
   OUTPUT('/yourpath/BILLS/IPSRA_TEST/FILE01_SAV')>
Compressing SAV02_IFS_TESTPKCS7* in SAVE Apps. Data mode
Add  SAV02_IFS_TESTPKCS7*  --  Deflating (63%)
Compressing /yourpath/BILLS/IPSRA_TEST/FILE01_SAV in BINARY mode
Add  /yourpath/BILLS/IPSRA_TEST/FILE01_SAV  --  Deflating (61%)
Smartcrypt Compressed 4 files in Archive /yourpath/bills/iPSRA_test/x3.zip
Smartcrypt Completed Successfully
```

Four files are stored in the archive. Two are the iPSRA files, and the other two are the outfiles in the commands.

iPSRA Example 6

The example below shows a restore error.

➔ **PKUNZIP ARCHIVE('/yourpath/bills/testsavx1.zip') TYPARCHFL(*IFS) FILES('SAVLIB01_DEDSAV01') TYPE(*EXTRACT) RSTIPSRA(RSTOBJ OBJ(TESTFILE) SAVLIB(TESTLIB) DEV(RSTTEST) OBJTYPE(*FILE) RSTLIB(DEDNEW) OUTPUT(*PRINT)) PASSWORD('bills00000')**

```
UNZIP Archive: /yourpath/bills/testsavx1.zip
Archive Comment:"Smartcrypt for IBM i"
Searching Archive /yourpath/bills/testsavx1.zip  for files to extract
Command:<RSTOBJ OBJ(TESTFILE) SAVLIB(TESTLIB) OBJTYPE(*FILE)
RSTLIB(DEDNEW) OUTPUT(*PRINT)>
Extracting file SAVLIB01_DEDSAV01
Inflating *iPSRA:SAVLIB01_DEDSAV01 iPSRA File
Restore Operation encountered an error.  See Job Log of PKZRSTA restore
job for further details.
SecureUNZIP  extracted     0 files
SecureUNZIP found     1 file(s) in Error


                        Additional Message Information

 Message ID . . . . . . :   AQZ1007
 Date sent  . . . . . . :   08/09/09      Time sent  . . . . . . :   09:54:57

 Message . . . . :   Restore Operation encountered an error.  See Job Log of
   PKZRSTA restore job for further details.

 DSPSPLF FILE(QPJOBLOG) JOB(019721/USER1/PKZRSTA) for job log and
   detail on why the restore operation failed.  Possible problem may be that
   some or all of the objects may not have been restored due to some restore
   setting.

Since OUTPUT(*PRINT) was in effect you could view the restore output:

*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....
 5722SS1 V5R3M0 040528                 RESTORE OBJECT INFORMATION
 OBJECT NAME  . . . . :   TESTFILE
 SAVE LIBRARY . . . . :   TESTLIB
 OBJECT TYPE  . . . . :   *FILE
 SAVE FILE NAME . . . :   QANE019357
 SAVE FILE LIBRARY  . :   QTEMP
 LABEL  . . . . . . . :   *SAVLIB
 OPTION . . . . . . . :   *ALL
 MEMBER OPTION  . . . :   *MATCH
 SAVE DATE/TIME . . . :
```

```
 ALWOBJDIF. . . . . . :   *NONE
 RESTORE LIBRARY  . . :   DEDNEW
 RESTORE ASP  . . . . :   *SAVASP
Specified file for library TESTLIB not found.
                          * * * * * E N D  O F  L I S T I N G * * * * *

Or from the DSPSPLF FILE(QPJOBLOG) JOB(019721/USER1/PKZRSTA) the job log will
show the actual IBM Restore error messages:

CPF3806    Diagnostic          20   08/09/09  09:54:57.125128  QANESERV     QSYS
 From module . . . . . . . . :   QANESERV
 From procedure  . . . . . . :   QaneSendPgmMsg__FP14qanec_CTLBLK_tPcT2iN24
 Statement . . . . . . . . . :   19
 To module . . . . . . . . . :   QP0ZPCPN
 To procedure  . . . . . . . :   InvokeTargetPgm__FP11qp0z_pcp_cb
 Statement . . . . . . . . . :   187
 Message . . . . :   Objects from save file QANE019357 in QTEMP not restored.
   Cause . . . . . :   The library name in the save file does not match the
   library name that you specified in the SAVLIB parameter. Recovery  . . . :
   Use the DSPSAVF command to display the save file and to determine the
   library from which the objects were saved. Specify the correct library in
   the SAVLIB parameter and try the command again.

CPF3780    Diagnostic          30   08/09/09  09:54:57.125152  QANESERV     QSYS
 From module . . . . . . . . :   QANESERV
 From procedure  . . . . . . :   QaneSendPgmMsg__FP14qanec_CTLBLK_tPcT2iN24
 Statement . . . . . . . . . :   19
 To module . . . . . . . . . :   QP0ZPCPN
 To procedure  . . . . . . . :   InvokeTargetPgm__FP11qp0z_pcp_cb
 Statement . . . . . . . . . :   187
 Message . . . . :   Specified file for library TESTLIB not found.
   Cause . . . . . :   The data in the save file or on the tape, diskette or
   optical volume did not match the specified parameters. Recovery  . . . :
   See the previously listed messages. If the restore operation is from
   diskette, tape or optical, display the contents of the volume using the
   appropriate display command specifying the DATA(*SAVRST) parameter. If the
   restore operation uses a save file, display the contents of the save file
   (DSPSAVF command). Correct any errors and then try the request again.
```

iPSRA Example 7

The example below shows how the save information is depicted for an object that was saved with PKZIP iPSRA and UPDHST(*YES) for the save command. Notice that the save file shows the save library of QTEMP and the save file as QANExxxxxx. This is an internal representation of the save API process. The device type will show as a save file.

➔ DSPOBJD OBJ(DED) OBJTYPE(*LIB) DETAIL(*FULL)

```
                        Display Object Description - Full
                                                                Library 1 of 1
 Object . . . . . . . :   DED             Attribute  . . . . . :   TEST
   Library  . . . . . :     QSYS          Owner  . . . . . . . :   WSS
 Library ASP device . :   *SYSBAS         Primary group  . . . :   *NONE
 Type . . . . . . . . :   *LIB

 Save/Restore information:
   Save date/time . . . . . . . . . . :   08/10/09  11:16:41
   Restore date/time  . . . . . . . . :   08/10/09  09:31:26
   Save command . . . . . . . . . . . :   SAVLIB
   Device type  . . . . . . . . . . . :   Save file
   Save file  . . . . . . . . . . . . :   QTEMP/QANE020372
```

7 PKZIP Command

PKZIP Command Summary with Parameter Keyword Format

To compress data from the IBM i OS command prompt screen, the command format is simply: ***PKZIP***. There also is a ***PKZSPOOL*** command which is the same command as PKZIP, but has the parameter TYPFL2ZP set to *SPL for spool files. The parameters are also re-sequenced to give preference to parameters dealing specifically with spool files.

The command prompt screen is displayed when Enter or PF4 is pressed. The parameter keywords are displayed on this screen, together with the available keyword options. If the command and parameter keywords are entered together on the command line the required format is:

PKZIP keyword1(option) keyword2(option) . . . keyword*n*(option)

Keywords are demarcated by spaces. The keyword "ARCHIVE" is the only positional keyword where the keyword is not required. Whenever the word "path" is used, its meaning depends on the file system that is being used. If IFS is used, path refers to the open system true path type. If the library systems or *DB is used, path means library/file and then the file name refers to the member name.

```
ADVCRYPT(  {ZIPSTD}                    )
               {AES128}          (Smartcrypt Only)
               {AES192}          (Smartcrypt Only)
               {AES256}          (Smartcrypt Only)
               {3DES}            (Smartcrypt Only)
               {DES}             (Smartcrypt Only)
               {RC4_128}         (Smartcrypt Only)
               {AE_2}            (Smartcrypt Only)
               {CAST5}           (PKPGPZ Only)

ARCHIVE(   Archive Zip File name with path       )
    Archive to create              { archive file name with path }
    Optional Input Archive File    { archive file name with path }
    Output Archive File Disposition {*DEFAULT}
                                   {*PROTECT}
                                   {*OVERWRITE}


ARCHTEXT(  {*NONE}                )
               Archive File Text description

AUTHCHK(   Authenticators         )           (Smartcrypt Only)
```

```
           Authenticate Type          {*FILE}
                                      {*ARCHIVE}
                                      {*ALL}
           Lookup Type                {*DB }
                                      {*LDAP}
                                      {*FILE}
                                      {*MBRSET}
                                      {*INLIST}
                                      {*SPONSOR} (Smartcrypt Partner (write mode)
               only)
           Recipient                  {Recipient String}
           Passphrase (if Private)   {Certificate passphrase}
           Required                   {*RQD }
                                      {*OPT}

AUTHPOL   (  Authenticate Filters:  )        (Smartcrypt Only)
Validate Level        { *SYSTEM }
                      {*WARN }
                      {*VALIDATE}
                      {*REQUIRED}
Validate Level        { *NONE }
                      {*ARCHIVE }
Filters               {*SYSTEM }
                      {*ALL}
                      {*NONE}
                      {*TAMPER}
                      {*TRUSTED}
                      {*EXPIRED}
                      {*REVOKED}
                      {*NOTAMPER}
                      {*NOTTRUSTED}
                      {*NOTEXPIRED}
                      {*NOTREVOKED}

COMPAT(    {*NONE}                ) Deprecated
               {*PK400}

COMPRESS(  Compression options    )
  Level        {*SUPERFAST}
               {*FAST}
               {*NORMAL}
               {*MAX}
               {*STORE}
               {*TERSE)
               {E1 thru E9}
  Method       {*DEFLATE32}
               {*DEFLATE64}
               {*BZIP2}
               {*STORE}
               {*TERSE)
CRTLIST(   {*NONE}                )
               path/filename
               {*SIMULATE}


CVTDATA(   External Pgm Conversion Extended Data)


CVTFLAG(   {*NONE}                )
               External Pgm Conversion Flags

CVTTYPE(   {*NONE}                )
               {*DROP}
               {*SUFFIX}

DATEAB(    mmddyyyy                     )
```

```
DATETYPE(   {*NO}                   )
                {*BEFORE}
                {*AFTER}

DBSERVICE(  ({*NO}                  )
                {*YES)

DELIM (     ({CRLF}                 )
                {CR }
                {LF }
                {LFCR }

DFTARCHREC({132}                    )
                {decimal number}

DIRRECRS(   {*NONE}                 )
                {*FULL}
                {*NAMEONLY}

ENTPREC(    Lookup Type; Recipient; Passphrase; Required )   (Smartcrypt Only)
            Lookup Type                 {*DB }
                                        {*LDAP}
                                        {*FILE}
                                        {*MBRSET}
                                        {*INLIST}
                                        {*SPONSOR} (Smartcrypt Partner (write mode)
                only)
            Recipient                   {Recipient String}
            Passphrase (if Private)     {Certificate passphrase}
            Required                    {*RQD }
                                        {*OPT}
ENCRYPOL    (  Encryption Filters:                )         (Smartcrypt Only)
            Validate Level         {*SYSTEM }
                                   {*WARN }
                                   {*VALIDATE}
            Filters                {*SYSTEM }
                                   {*ALL}
                                   {*NONE}
                                   {*TRUSTED}
                                   {*EXPIRED}
                                   {*REVOKED}
                                   {*NOTTRUSTED}
                                   {*NOTEXPIRED}
                                   {*NOTREVOKED}


EXCLFILE(   {*NONE}                 )
                path/filename

EXCLUDE(    file_specification 1,         )
                file_specification 2,
                file_specification n

EXTRAFLD(   {*YES}                  )
                {*NO}
                {*CENTRAL}
                {*LOCAL}
                {*BOTH same as *YES}


ERROPT(     {*END}                  )
                {*SKIP}
                {*WARN}

FACILITY (  Algorithm Facilities )                  (Smartcrypt V5R3M0 Only)
            Encryption:                 {*DFT}
                                        {PKSW }
                                        {IBMSW }
```

```
                                       {PKSW_IBMSW}
                                       {IBMSW_PKSW}
           Hashing:                   {*DFT }
                                       {PKSW }
                                       {IBMSW }
                                       {PKSW_IBMSW}
                                       {IBMSW_PKSW}

FILES(     file_specification 1,        )
               file_specification 2,
               file_specification n
           OR *COPY
FILESTEXT( {*NO}                  )
               {*ALL}
               {*NEW}
               {*UPDATE}

FILETYPE(  {*TEXT}                )
               {*BINARY}
               {*EBCDIC}
               {*FIXTEXT}
               {*DETECT}

FNE(       {Create FNE | Overwrite FNE}   )        (Smartcrypt Only)
               {*YES | *NO}

FTRAN(       {*ISO88591}          )
             {*INTERNAL}
              Member Name

GZIP(      {*YES}                 )
               {*NO}

IFSCDEPAGE({*NO}                  )
               Code-page

INCLFILE(  {*NONE}                )
               path/filename

MSGTYPE(   Outlist Details:       )
  Type         {*BOTH}
               {*PRINT}
               {*SEND}

  License Info {*NORMAL}
               {*SHORT}
               {*NONE}
               {*COPYRIGHT}

NSSRULES   (  NSS Process Settings:                     )   (Smartcrypt Only)
           NSS Classify Archive   {*SYSTEM }
                                  {*NO }
                                  {INACTIVE}
                                  {SECRET_SUITEB_REQPLUS}
                                  {SECRET_SUITEB_STRICT}
                                  {TOPSECRET_SUITEB_REQPLUS}
                                  {TOPSECRET_SUITEB_STRICT}


           NSS Check Archive State
                                  {*SYSTEM }
                                  {*NO}
                                  {*OPT}
                                  {*WARN }
                                  {*FAIL}


PASSWORD(  Archive Passphrase           )
```

```
PKOVRTAPF (  Archive Tape Overrides:                      )
           Tape Device           {*TAPF }
                                 { Tape Device }
           Tape File Label       {*TAPF }
                                 { Tape Header Label}
           Tape Sequence Nbr     {*TAPF }
                                 { 1-16777216 }
                                 {*END}
           File expiration date  {*TAPF }
                                 { Date}
                                 {*NONE }
                                 {*PERM}
           End Of Tape Option    {*TAPF }
                                 {*LEAVE}
                                 {*REWIND}
                                 {*UNLOAD}
           Shadow Dir File       {*CSDF}
                                 {*NO}


SELFXTRACT (   {*MAINTAIN}        )
               {*REMOVE}
               {WINDOWS}
               {AIX}
               {HP_UNIX}
               {SUN_UNIX}
               {LINUXINTEL}
               {SF2WINC}
               {SF2AIX}
               {SF2HP}
               {SF2SUN}
               {SF2LNX2I}
               {SFAWINC}
               {SFAWING}
               {SF6AIX}
               {SF6HP}
               {SF6SUN}
               {SF6LNX2I}

SFUSER (   {*CURRENT}             )
               {user id 1}
               {user id 2}
               {user id 5)
               {*ALL}

SFQUEUE (  {*ALL}                 )
               {Library/OUTQ }

SFFORM (   {*ALL}                 )
               {*STD}
               {Spool File Form Type }

SFUSRDTA ( {*ALL}                 )
               {Spool File User data}

SFSTATUS ( {*ALL}                 )
               {*READY}
               (*HELD }
               {*CLOSED}
               {*SAVED }
               {*PENDING}
               {*DEFERRED}

SFJOBNAM ( {blanks }                    )
               {*}
               {Job-name//User-Name/Job Number}
```

```
SFTARGET ( {*SPLF}                )
              {*TEXT}
              {*TEXT1}
              {*TEXT2}
              {*TEXTFC}
              {*PDF}
              {*PDFLETTER}
              {*PDFLEGAL}


SFTGFILE ( {*GEN1}                )
              {*GEN2}
              {*GEN1P}
              {path/filename }

SIGNERS(   Signer         )                        (Smartcrypt Only)
           Signing Type                {*FILE}
                                       {*ARCHIVE}
                                       {*ALL}
           Lookup Type                 {*DB }
                                       {*LDAP}
                                       {*FILE}
                                       {*MBRSET}
                                       {*INLIST}
           Recipient                   {Recipient String}
           Passphrase (if Private)    {Certificate passphrase}
           Required                    {*RQD }
                                       {*OPT}

SIGNPOL   (  Signing Filters:                    )   (Smartcrypt Only)
           Validate Level         {*SYSTEM }
                                  {*WARN }
                                  {*VALIDATE}
           Filters                {*SYSTEM }
                                  {*ALL}
                                  {*NONE}
                                  {*TRUSTED}
                                  {*EXPIRED}
                                  {*REVOKED}
                                  {*NOTTRUSTED}
                                  {*NOTEXPIRED}
                                  {*NOTREVOKED}

           Signing Hash           {*SYSTEM }}
                                  {*SHA1}
                                  {*MD5}
                                  {*SHA256}
                                  {*SHA384}
                                  {*SHA512}


STOREPATH(     {*NO}                    )
              {*YES}
              {*REL }
              {*NOROOT}



SPLFILE (  {*ALL}                 )
              {Spool File Name }

SPLNBR (   {*ALL}                 )
              {*LAST}
              {Spool File Number 1-9999}

TMPPATH(   {*CURRENT}             )
              Temporary Path
```

```
TRAN(        {*ISO88591}         )
             {*INTERNAL}
              Member Name

TYPARCHFL(  Archive Type File      )
   Type        {*DB}
               {*IFS}
               {*TAP}
               {*XDB}

  Check ZIP64  {*NONE}
               {*WARN}
               {*FAIL}

TYPE(       *ADD                   )
               {*UPDATE}
               {*FRESHEN}
               {*MOVEA}
               {*MOVEF}
               {*MOVEU}
               {*DELETE}


TYPFL2ZP(      {*DB}                      )
               {*IFS}
               {*IFS2}
               {*DBA}
               {*SPL}

TYPLISTFL(     {*DB}                      )
               {*IFS}

VERBOSE(    {*NORMAL}              )
               {*NONE}
               {*ALL}
               {*MAX}

VPASSWORD(  Archive Verify Passphrase     )
```

PKZIP Command Keyword Details

ADVCRYPT

ADVCRYPT(ZIPSTD|AES128|AES192|AES256|DES|3DES|RC4_128|AE_2|CAST5)

Note: PKZIP for IBM i only supports *NONE and ZIPSTD options.

When a ZIP action is requested to save a file in an archive, and a passphrase is provided, ***Smartcrypt for IBM i*** will use an encryption method to protect the data.

This command value specifies which algorithm to employ.

Possible encryptions are:

ZIPSTD	This algorithm is the original algorithm used in PKZIP 2.x products and is compatible with other PKZIP 2.04g products that support standard encryption. Unless the installation defaults module has been tailored differently,

	this is the default value for ***Smartcrypt***[i] if you choose to encrypt a file.
***NONE**	No Encryption
AES128	Advanced Encryption Standard 128-bit key algorithm, also known as Rijndael.
AES192	Advanced Encryption Standard 192-bit key algorithm, also known as Rijndael.
<u>AES256</u>	Advanced Encryption Standard 256-bit key algorithm, also known as Rijndael. This is the default value for ***Smartcrypt for IBM i***.
DES	Data Encryption Standard.
3DES	Triple Data Encryption Standard.
RC4_128	RC4 is a stream cipher created by RSA.
AE_2	A passphrase-based symmetric key algorithm intended for distribution of ZIP archives to operating environments where this is the preferred method (WinZip).
CAST5	A Smartcrypt implementation of the CAST5 algorithm used only for OpenPGP files.

Usage Notes:

PKUNZIP will detect automatically which encryption method was specified during the ZIP process and operate accordingly.

During a PKZIP (ZIP) run, only one encryption method may be specified, and that method will be used for each file that is operated on.

By executing PKZIP at different times, various files within the archive may be saved with differing levels (and types) of encryption. That is, some files may not be protected at all, while others may have different methods and/or passphrases.

A “+” character is shown in a view to indicate standard encryption protection is used for a file.

A “!” character is shown in a View to indicate advanced encryption (AES) protection is used for a file.

OPENPGP formats only supports AES128, AES192, AES256, 3DES, and CAST5 .

ARCHIVE

ARCHIVE(archive name, optional input archive name, output archive Disp)

```
Archive Zip File:
       Archive Name  . . . . . .   _________________________________________
       (Optional) Input  . . . .   *NONE____________________________________
       Output Archive Disp.  . .   *DEFAULT       *PROTECT, *OVERWRITE...
```

Or

> **ARCHIVE('/yourpath/mypath/myarch.zip')**
> **ARCHIVE('/yourpath/mypath/myarch.zip' *NONE *OVERWRITE)**
> **ARCHIVE('/yourpath/mypath/myarch.zip' '/yourpath/mypath/oldarch.zip')**
> **ARCHIVE('MYLIB/MYARCH/NEWZIP1' 'MYLIB/MYARCH/OLDZIP0' *PROTECT)**

This parameter specifies the archive files for output and/or input. Currently there are 3 entries for the ARCHIVE parameter (Archive File to create, Optional Input Archive file, and output file disposition).

Archive to create (archive file name with path)

Specifies the path/file name or the library/file name of the ***Smartcrypt***^{i} archive to be processed. If the file exists, PKZIP will overwrite the file, otherwise PKZIP will create the file for you. Depending on which file system you choose, the path or library must exist. This is a required parameter.

Optional Input Archive File (archive file name with path)

Specifies the path/file name or the library/file name of an archive the will be used as input. This parameter provides the ability to have an input archive to update but this archive is preserved and not updated. The files in the archive will be copied to the new updated archive along with any new file selections. If an existing archive is to be updated with the same archive name then using the "archive to create" parameter is only required.

Output Archive File Disposition (*DEFAULT| *PROTECT|*OVERWRITE)

Specifies the output archive's disposition if it exist.

<u>*DEFAULT</u>	This option provides backward compatibility to version prior to 8.2. If no input archive is provided, this option is set to *OVERWRITE. If an Inputted archive is provided then this option will be set to *PROTECT.
***PROTECT**	If the output archive file exist, do not overwrite the archive and fail the run.
***OVERWRITE**	If the output archive exist, then overwrite the archive with the new or updated archive.

NOTE: archive file name with path:

The format of “archive file name with path” depends on whether you will be using the archive file in the library file system, or the IFS (Integrated File System).

See parameter TYPARCHFL for file system type information.

Library File System
Format is *library/file(member)*. If *member* is omitted, it will be created with the file name. If the file is not found, it will be created with a default length specified in parameter DFTARCHREC (which has a default of 132). If you want to create a file manually to use a larger record length, create it with no members and with the parameter MAXMBRS with *NOMAX, or with a high excepted limit. If the Library is not specified, the file name will be searched using *LIBL. If the file name is not found, the file will be created in the users *CURLIB. If a library is specified and does not exist, PKZIP will create the library.

Integrated File System (IFS)
Open system path followed by the archive file name. The path and file name can up to 256 characters and may contain embedded spaces.

ARCHTEXT

ARCHTEXT(*NONE| Archive File Text description)

Specifies text that will be stored in the archive as the archive's file comment.

***NONE**	No new archive comment will be stored.
***DEFAULT**	The default PKWARE comment will be stored.
***CLEAR**	Clear any comment that may be stored in an archive.
Archive File Text description	Up to 255 characters that are stored as the archive's file comments.

AUTHCHK

Requires Smartcrypt

```
Authenticator Certificates:
    File/Archive   . . . . . . .   * ARCHIVE     *ARCHIVE
    LookUp Type  . . . . . . . .   *DB           *DB, *LDAP, *FILE, *MBRSET...
    Authenticator  . . . . . . .   ______________________________
    Passphrase (If Private)  . . . ______________________________
    Required . . . . . . . . . .   *RQD          *RQD, *OPT
               + for more values  _
```

Or

```
AUTHCHK((*ARCHIVE *MBRSET
   'pkwareCertAdmin04.pfx'  (passphrase) *RQD))
AUTHCHK((*ARCHIVE *FILE
   '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer'  () *RQD))
AUTHCHK((*ARCHIVE *DB
   'EM=bill.somebody@pkware.com'  () *OPT))
AUTHCHK((*ARCHIVE *INLIST 'ATEST/INLIST(ENGNEER1)' *N)
```

This parameter specifies that digital signature authentication processing should be performed for specific signers. Separate authentication processing may be specified for either the archive central directory or files by using multiple commands. Optionally, specific signers may be specified to authenticate against. This parameter is used in conjunction with the AUTHPOL parameters and its settings.

It is possible that more than one certificate may be returned for a single common name or email search. As a result, each one will be added to the list of validating sources.

When no specific certificates are requested, any signatories found in the archive are validated in accordance with the systems or current AUTHPOL Filters policy settings.

There are five options for AUTHCHK.

Authenticator Type File/Archive **(*ARCHIVE)**

Indicates the type of archive authentication to do. If the lookup type is *INLIST, then this option will be ignored and will pick up from the records in the inlist file.

- *ARCHIVE - The archive directory will be authenticated with this authenticator.

Lookup Type **(*DB |*FILE |*LDAP |*MBRSET |*INLIST |*SPONSOR)**

The lookup type would be the type of authenticator search to be used for the authenticator string to look up the public key.

- *DB - The authenticator string is defined to search using the certificate locator database to access the digital certificate.
- *FILE - The authenticator string is defined to read a specific file in a specific path in the IFS in order to access the digital certificate.
- *LDAP - The recipient string is defined to search using the LDAP server to access the digital certificate.
- *MBRSET - The authenticator string is defined to read this specific file from the enterprise public certificate store to access the digital certificate.
- *INLIST- The authenticator string defines a specific file that will contain one to many AUTHCHK. The TYPLISTFL parameter must specify the file type for the inlist.
- If lookup type is *SPONSOR, the authenticator string is Sponsor Auth file stored in the '.../Sponsor/Auth' folder. If the authenticator string is all numeric, the name will automatically be formatted as A0000000.p7, assuming that the number is the sponsor ID number. *(Write mode only)*

Authenticator (The authenticator string name)

The authenticator string format depends on what was specified for the lookup type.

- If lookup type is *DB, the authenticator string will either be an email address or the common name of the certificate. This depends on the configuration setting in PKCFGSEC parameter CERTDB. To override the default selection mode, you can prefix the string with EM= for email, or CN= for the common name.

For example:

AUTHCHK((*ARCHIVE *DB 'CN=bill somebody' () *RQD))

- If lookup type is *FILE, the authenticator string is defined to read a specific file in a specific path of the IFS. This file should be a public key X.509 file or public key X.509 certificate with a private key file.

For example:

AUTHCHK((*ARCHIVE *FILE
'/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer' () *RQD))

The digital certificate file 'pkwareCertAdmin04.cer' will be in the full path '/yourpath/PKWARE/Cstores/public'.

- If type is *LDAP, the authenticator string will either be an email address or the common name of the certificate depending on the search mode configuration setting in PKCFGSEC parameter LDAP. To override the default selection mode, you can prefix the string with EM= for email address, or CN= for the common name.

For example:

```
AUTHCHK ((*ARCHIVE *LDAP  'bill.somebody@pkware.com'  () *RQD)
        (*ARCHIVE  *LDAP  'CN=bill somebody'  () *OPT))
```

- If lookup type is *MBRSET, the authenticator string is defined to read a specific file from the public certificate store and/or the private certificate store of the IFS. This file should be a public key X.509 file or public key X.509 certificate with a private key file.

For example:

AUTHCHK((*ARCHIVE *MBRSET 'pkwareCertAdmin04.cer' () *RQD))

The digital certificate file 'pkwareCertAdmin04.cer' will be in the full path of the public certificate store defined in the enterprise security configuration public store (parameter CSPUB). If a passphrase is included, the file is searched for in the enterprise security configuration private store (parameter CSPRIV).

- If lookup type is *INLIST, the authenticator string defines a full file name of an input list file that contains records of AUTHCHK shortcut parameters. The type of file will exist in the QSYS library file system if TYPLISTFL(*DB) is set and will be a path file name in the IFS if TYPLISTFL(*IFS) is set. The format of the AUTHCHK shortcut parameters are defined below in the *INLIST usage section.

Passphrase

This designates the passphrase that is *required* for a private key certificate with a private key (PKCS#12 file). When a value is specified, the target must be an X.509 PKCS#12 public key certificate with the private key.

The PASSWORD value may contain blanks and is delimited by the closing right parenthesis ")" of the signing command.

Required **(*RQD|*OPT|<u>*SAME)</u>**

If *RQD, then this authenticator *must* be found during the selection, and the certificate *must* be a valid certificate with a private key, or the ZIP/UNZIP run will fail.

Usage Notes:

Passphrases are masked out in all output displays.

A local certificate store configuration is required to complete the TRUST processing of this command.

Processing is terminated if none of the requested certificates can be accessed, regardless of the “R” required flag. If multiple requests are made and at least one signature is found, processing continues normally.

For inlist that contains a passphrase to open a private certificate, make sure that the security is sufficient to only allow the owner of the certificate to have read access. Otherwise this would leave a security hole where other users could browse the passphrase.

***INLIST Usage:**

If *INLIST is defined on the AUTHCHK parameter, then the authenticator filed will be a file that Smartcrypt will read to include the authenticator. The format is very similar to the AUTHCHK parameter described above except that each line authenticator starts with “{AUTHCHK=” and is terminated by the “}” character, with the semi-colon “;” as a separator for each entry.

{AUTHCHK=Authenticator Type, Lookup Type; Authenticator; Passphrase; Required}

Authenticator *Type*	See Authenticator Type in AUTHCHK
Lookup Type	See Lookup Type in AUTHCHK excluding the INLIST
Authenticator	See Authenticator in AUTHCHK.
Passphrase	See Passphrase in AUTHCHK.
Required	See Required in AUTHCHK, but use RDQ for *RQD and OPT for *OPT.

Examples:

Sample 1: tstauth_db1.inlist.

```
{AUTHCHK=ARCHIVE;DB;EM=PKTESTDB4@nowhere.com;;RQD}
```

Sample 2: tstauth_mb2.inlist.

```
{AUTHCHK=ARCHIVE;MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

AUTHPOL

Requires Smartcrypt

```
Authenticate Filters:
    Validate Level     . . . . .   *SYSTEM       *VALIDATE, *WARN, *NONE...
    Validate Type      . . . . .   *ARCHIVE      *ARCHIVE, *NONE
       Filters . . . . . . . . .   *SYSTEM       *SYSTEM, *ALL, *NONE...
               + for more values
```

Or

AUTHPOL(*WARN *ARCHIVE (*SYSTEM))
AUTHPOL(*WARN *FILE (*NOTTRUSTED))
AUTHPOL(*SYSTEM *ALL (*ALL *NOTEXPIRED))

This parameter defines the processing options and filters that should apply if a signed file or signed archive is encountered.

Validate Level (*VALIDATE |*WARN |*REQUIRED |<u>*SYSTEM</u>)

The validate level specifies the type of authentication processing that should take place if a signed archive is encountered. The default is *SYSTEM and, unless it is modified, Smartcrypt will use the enterprise setting from PKCFSEC.

- *VALIDATE – Indicates that when authentication takes place and a failure occurs based on the filters, the run will be considered a failure and the message issued when the job terminates will indicate one or more errors during the run.
- *WARN - Indicates that when authentication takes place and a failure occurs, the failure is only to be considered a warning. The messages at the end of the run will not consider any failed authentications as errors.
- *REQUIRED – Indicates that authentication *must* take place and that, if any failure occurs based on the filters, the run will be considered a failure, and the message issued when the job terminates will indicate one or more errors occurred during the run. If the archive has not been signed, then an error will be issued.
- <u>*SYSTEM – </u>Indicates the authentication processing that is set in the environmental setting will be used.

Validate Type (*ARCHIVE <u>|*NONE</u>)

The validate type specifies that archive authentication should take place if an archive has been signed. The default is *NONE and anything other than *NONE requires the Enhanced Encryption Feature.

- *ARCHIVE - Indicates that only a signed archive will be authenticated.
- *NONE - Indicates no authentication will take place even though a file or archive has been signed.

Filters (<u>*SYSTEM</u> |*ALL |*NONE |*TAMPER |*TRUSTED |*EXPIRED |*REVOKED |*NOTAMPER |*NOTTRUSTED |*NOTEXPIRED |*NOTREVOKED)

The authentication filter policies settings are defined in the enterprise security file supplied by the Smartcrypt administrator (See PKCFGSEC). These global policy settings can be revised with sub-parameter values. The variables are cumulative from the global setting.

- **<u>*SYSTEM</u>** – All filter policies are from the global settings.
- ***ALL** - This sub-parameter activates all levels of authentication. If followed by negating sub-levels, then all but those negating levels are activated. For example: *ALL NOTEXPIRED means that expired certificates will not cause an authentication error, but TRUST and TAMPERCHECK must both be satisfied.
- ***NONE** – Will negate all the policies.
- ***TAMPER** – This sub-parameter signifies that a verification of the data stream should be done against the digital signature.
- ***TRUSTED** – This sub-parameter signifies that the entire certificate authority chain must be validated. This includes locating the root (self-signed) certificate on the local system.
- ***EXPIRED** – This sub-parameter signifies that certificate date range validation should be performed on the certificates (including the certificate authority chain). Although the term "expired" is used, a certificate that has not yet reached its valid data range specification will fail.
- ***REVOKED** - A certificate owner may request that the issuing certificate authority declare a certificate to be revoked and thereby no longer consider that certificate to be valid. The authentication operation will fail if any of the certificates in the trust chain are found to have been revoked, or if the revocation status could not be determined
- ***NOTAMPER** – Negates the *TAMPER filter.
- ***NOTTRUSTED** – Negates the *TRUSTED filter.
- ***NOTEXPIRED** - Negates the *EXPIRED filter.
- ***NOTREVOKED** – Negates the *REVOKED filter.

<u>COMPAT</u>

COMPAT(<u>*NONE</u>|*PK400)   Deprecated

Specifies that PKZIP will create and store extended data field information in another supported format or previous version. At this time, only "PKZIP Version 4.0 for OS/400" is supported.

The allowable values are:

<u>*NONE</u>	The extended data fields will be in ***Smartcrypt***^{i} versions 5.0 and above formats.
***PK400**	The extended data fields will output to the archive in the format used by “PKZIP Version 4.0 for OS/400” product. This option should be used if the archive file will be extracted by “PKZIP Version 4.0 for OS/400” and the attributes are required to create the files. The files can be extracted without this option, but the files may have to be manually created in order to have the proper attributes (such as record length and text descriptions).

COMPRESS

```
Compression:
              Level . . . . . .   *SUPERFAST    *SUPERFAST, *FAST, *NORMAL...
              Method  . . . . .   *DEFLATE32    *DEFLATE32, *DEFLATE64...
```

Or

COMPRESS(*FAST *DEFLATE64)
COMPRESS(E1 *DEFLATE32)
COMPRESS(*STORE)

This parameter specifies the speed and compression level when zipping a file. Currently there are 2 entries for the COMPRESS parameter (Level and Method).

Compression Level (<u>*SUPERFAST</u>| *FAST| *NORMAL|*MAX|*STORE|*TERSE |E0 thru E9)

The compression level option specifies a compression level and speed to be used. The option works in conjunction with the compression method option and specifies a depth of compression using a sliding scale of values.

The allowable values are:

***FAST**	Fast selection provides ample compression at a fast rate. Same as E2.
<u>*SUPERFAST</u>	This is the default selection. This will compress in the fastest time, but will compress the files by the least amount. Same as E1.
***MAX**	This level provides the maximum compression possible, but will also take the longest in time to process. Same as E6.
***NORMAL**	The normal compression level provides good compression amount at a reasonable speed. Same as E3.
***STORE**	No compression. Store will also be used if the other methods tried result in a file larger than the original. Same as E0.
***TERSE**	This selection provides a terse compression algorithm provided with the IBM i by IBM as an API. This is much

faster but is less efficient than FAST, and can only be decompressed on the IBM i. Do not use this option if you wish to unzip the archive on another platform.

E0 thru E9	E0 thru E9 are custom levels that can be used to try and obtain the results based on your input files and desired time and compression results.

The following table shows the balance of degree of compression and speed of compression. The levels range from 0 (fastest speed with no compression) to 9 (highest level of compression, usually taking the longest amount of time and using the most processor time).

Synonym	***Level***	***Usage***
STORE, E0	0	No compression is performed.
SUPERFAST, E1	1	Compression Method: Deflate32 or Deflate64
FAST, E2	2	Compression Method: Deflate32 or Deflate64
NORMAL, E3	3	Compression Method: Deflate32 or Deflate64
E4	4	Compression Method: Deflate32 or Deflate64
E5	5	Compression Method: Deflate32 or Deflate64
MAXIMUM, E6	6	Compression Method: Deflate32 or Deflate64
E7	7	Compression Method: Deflate32 or Deflate64
E8	8	Compression Method: Deflate32 or Deflate64
E9	9	Compression Method: Deflate32 or Deflate64

Usage Notes:

- Compression levels 1 through 9 all work with Deflate32, Deflate64, and BZIP2 compression methods.
- “Maximum” is retained at level 4 to provide equivalent compression ratios with earlier releases. Higher levels may yield better compression ratios than previously obtained with “Maximum”.
- Compression results are data-stream dependent and produce non-linear results. When configuring a job for high volume processing, benchmarking results with sample file may be of value to find the best balance between compression ratio and resources (elapsed and processor time).
- In many cases, levels 8 and 9 do not produce significant compression results over level 7.
- When compression level is STORE, or E0, the compression method will be set automatically to store.
- When migrating from earlier releases of ***Smartcrypt**^{i}*, a difference in compression ratio/processor time can be expected for a given data stream and setting. Although internal settings have been tuned to produce similar results across the scale of levels, a specific level setting may not produce faster speeds or better compression for a data stream. If these criteria are of importance, then benchmarking should be performed to achieve the “best” fit results with the new algorithms.

- BZIP2 compression levels are associated with memory management for the algorithm. Higher numbers for the level may significantly impact the region requirements for the run. For more information about BZIP2 member management, see http://bzip.org/1.0.5/bzip2-manual-1.0.5.html#options.

Method (<u>*DEFATE32</u> |*DEFLATE64 |*BZIP2 |*STORE |*TERSE)

This option specifies the algorithm to be used when compressing a file during ZIP processing. The method works in conjunction with the compression level option to specify a depth of compression.

STORE performs no compression of the data. Deflate64 (using the same level control) will usually produce better compression with less processor time than Deflate32.

The allowable values are:

<u>*DEFLATE32</u>	Use the Deflate 32 algorithm.
***DEFLATE64**	Use the Deflate 64 algorithm.
***BZIP2**	Use the BZIP2 compression algorithm.
***STORE**	Store Data with no compression.
***TERSE**	Use the IBM Terse algorithm.

Usage Notes:

- When compression method is store is specified, the compression level will be set automatically to *STORE.
- The GZIP specification only supports Deflate32. When GZIP(*YES) is encountered, PKZIP will automatically switch from Deflate64 or STORE to Deflate32.
- Not all non-PKWARE "ZIP compatible" products in the market support the more advanced Deflate64 algorithm. If the intended target systems support Deflate64, then it may be chosen for the best compression/speed performance.

<u>CRTLIST</u>

CRTLIST(<u>*NONE</u>| *path/filename* | *SIMULATE)

Specifies that PKZIP will create an output file with a list of entries that would have been compressed based upon the selection criteria in the FILES and EXCLUDE parameters.

Use FILES and EXCLUDE to generate a listfile; use INCLFILE in a separate command to load the listfile.

See parameter TYPLISTFL for file system type.

<u>*NONE</u>	Default. No list file will be created.

path/filename
Enter the file path and name of the file to create. The layout depends on which file system you want to create the file in.

Library File System: The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

***SIMULATE** Will simulate the file selection and show the selection as a printed or message list instead of writing to a list file.

CVTDATA

CVTDATA(*External Program Conversion Extended Data*)

Specifies the extended data that is passed to the external program CVTNAME. When CVTFLAG is not *NONE, the contents of the parameter are passed to provide extended flexibility in controlling how the IBM i names are stored in the archive. The *System Administrator's Guide* contains more information on CVTNAME.

External Program Conversion Extended Data
Specify up to 255 bytes of unedited data which is passed to the exit program CVTNAME to assist in controlling the program logic.

CVTFLAG

CVTFLAG(<u>*NONE</u>| *Conversion Flags*)

Specifies the flags passed to the external program CVTNAME. These are used to control how the IBM i names are stored in the archive. The *System Administrator's Guide* contains more information on CVTNAME.

<u>*NONE</u>	Conversion exit is not active.
Conversion Flags	Specify a five-byte flag that is passed to the exit program CVTNAME to control the program logic. If the name passed back is blank, then conversion is referred back to the setting of the CVTTYPE parameter.

CVTTYPE

CVTTYPE(*NONE|*DROP|<u>*SUFFIX</u>)

Specifies how the IBM i library and file names are stored in the archive. Since the length of the library name, file name, and member name can each be up to 10 characters, and MS/DOS format requires a maximum of 8 characters with an optional extension, this option allows name compatibility.

The allowable values are:

<u>*SUFFIX</u>	This forces any IBM i name with more than 8 characters to create a name of 8 characters and a period(.), followed by characters 9 and 10 to be considered an extension to suffix.
***NONE**	This leaves the IBM i name as the archive name.
***DROP**	This forces any IBM i name with more than 8 characters, to drop characters 9 thru 19.

<u>DATEAB</u>

DATEAB(mmddyyyy)

Used with DATETYPE parameter, DATEAB specifies the date to be used to compare with the files latest modification date for file selection. The format is mmddyyyy, where “mm” is a valid month (01-12), “dd” is valid day of the month, and “yyyy” is the four digits of the year (2001).

<u>DATETYPE</u>

DATETYPE(<u>*NO</u>|*BEFORE|*AFTER)

Specifies if PKZIP should select files based upon a file modification date.

The allowable values are:

<u>*NO</u>	No date selection will take place.
***BEFORE**	Files with a modification date before the date in DATETYPE will be selected.
***AFTER**	Files with a modification date on or after the date in DATETYPE will be selected.

<u>DBSERVICE</u>

DBSERVICE (<u>*NO</u>|*YES)

Specifies if the IBM i special database extended file attributes describing the database file, fields and keys are to be store in the archives. This will force the option EXTRAFLD(*YES). The database will also be stored in binary mode. This mode can produce larger archive files.

The allowable values are:

<u>*NO</u>	Does not store database extended services attributes.
***YES**	Stores the database extended service attributes in the archive file and treat non-SAVF as a database.

DFTARCHREC

DFTARCHREC(<u>132</u>|Record Length)

Specifies the record length to use when creating an archive file in the QSYS library system. If the TYPARCHFL parameter is *DB or *XDB, and the archive file does not exist, the archive file will be created with the record length specified in this parameter.

Note: A large record length will leave a high residual number if only one byte is use in the last record.

The allowable values are:

<u>132</u>	Default is record length of 132 to match previous versions.
Record Length	A decimal number from 50 to 32000.

DELIM

DELIM(<u>CRLF</u> |CR |LF |LFCR)

When compressing a text file (not binary), the DELIM parameter specifies what characters are to be appended at the end of records to serve as delimiters. The delimiter is removed from the record when it is decompressed.

The allowable values are:

<u>CRLF</u>	This is the default selection. Specifies for ***Smartcrypt***^{i} to use the default delimiter CR-LF (x'0D0A') at the end of each text record.
CR	Appends an ASCII carriage return (hex 0D).
LF	Appends an ASCII line feed character (hex 0A).
LFCR	Appends an ASCII line feed character and a carriage return (hex 0A0D).

Note that transfers of MS-DOS records use a CRLF for a delimiter, while UNIX records use a LF.

DIRNAMES

DIRNAMES(<u>*YES</u>|*NO) Deprecated

Specifies to store directories as an entry. This is valid only for files in IFS.

<u>*YES</u>	Store the directories as entries in the archive.
***NO**	Do not store directories as an archive entry.

DIRRECRS

DIRRECRS(*NONE|*FULL|*NAMEONLY)

IFS only. Specifies whether to search recursively through directories for file selection, or only search the current, specified directory.

The allowable values are:

***NONE**	Search only the current, specified directory.
***FULL**	Search through all directories by starting with the current, specified directory for selected files. If *FULL is used, and * is for file selections, all files found in all directories below the current directory will be selected.
***NAMEONLY**	To be considered a hit, the full path and file name must match the selection statements exactly.

ENTPREC

Requires Smartcrypt

```
Encryption Recipients       :
     LookUp Type  . . . . . . . .   *DB      *DB, *LDAP, *FILE...
     Recipient  . . . . . . . . . ______________________________________

     Passphrase (If Private)  . . ______________________________________
     Required . . . . . . . . . .   *RQD           *RQD, *OPT
                + for more values   _
```

Or

ENTPREC((*MBRSET 'pkwareCertAdmin04.cer' () *RQD))
ENTPREC((*FILE '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer' () *RQD))
ENTPREC((*FILE '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.pfx' ('mypassphrase') *RQD))
ENTPREC((*DB 'EM=bill.Somebody@pkware.com' () *RQD))
ENTPREC((*LDAP 'EM=bill.Somebody@pkware.com' () *RQD))
ENTPREC((*INLIST 'ATEST/INLIST(ENGNEER1)' *N))

The encryption recipient parameter defines one to many Recipients which is to be included for the ZIP process. This parameter allows 1-4 types of certificate searches to take place along with providing the ability for an include file that may contain the recipients.

The specification of this recipient ENTPREC parameter, triggers encryption to take place during ZIP processing utilizing the found recipients along with any passphrase that may be entered.

There are four entries for the ENTPREC parameter (lookup type, recipient, passphrase, and required).

Lookup Type (*NONE |*DB |*LDAP |*FILE |*MBRSET |*SPONSOR |<u>*SAME)</u>

The Lookup type would be the type of recipient search that will be used for the recipient string.

- *DB - The recipient string is defined to search using the Certificate Locator Database to access the digital certificate.
- *LDAP - The recipient string is defined to search using the LDAP server to access the digital certificate.
- *FILE - The recipient string is defined to read a specific file in a specific path in the IFS in order to access the digital certificate.
- *MBRSET - The recipient string is defined to read this specific file from the enterprise public certificate store to access the digital certificate.
- *INLIST - The recipient string defines a specific file that will contain 1 to many recipients.
- *SPONSOR - The recipient string is the encryption recipient file for a sponsoring partner. This applies only for SecureZIP Partner Read mode.

Recipient (The recipient string name)

The recipient string format depends on what was specified for the Lookup type.

- If type is *DB: The recipient string will either be an email address or the common name of the certificate. This depends on the configuration setting in PKCFGSEC parameter CERTDB. To override the default selection mode, you can prefix the string with EM= for email address or CN= for the common name.

  For example:

  ```
  ENTPREC((*DB    ‘bill.Somebody@pkware.com'  () *RQD)
       (*DB    ‘CN=bill Somebody'  () *RQD)
       (*DB    ‘EM=bill.Somebody@pkware.com'  () *RQD))
  ```

- If type is *LDAP: The recipient string will either be an email address or the common name of the certificate depending on the search mode configuration setting in PKCFGSEC parameter LDAP. To override the default selection mode, you can prefix the string with EM= for email address or CN= for the common name.

  For example:

  ```
  ENTPREC((*LDAP    ‘bill.Somebody@pkware.com'  () *RQD)
         (*LDAP    ‘CN=bill Somebody'  () *OPT)
         (*LDAP    ‘EM=bill.Somebody@pkware.com'  () *RQD))
  ```

- If type is *FILE: The recipient string is defined to read a specific file in a specific path of the IFS. This file should be public-key X.509 file or private-key X.509 certificate file.

  For example:

  ```
  ENTPREC((*FILE '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer'  ()
  *RQD))
  ```

The digital certificate file ‘pkwareCertAdmin04.cer’ will be in the full path ‘/yourpath/PKWARE/Cstores/public’.

- If type is *MBRSET: The recipient string is defined to read a specific file from public certificate store / private certificate store of the IFS. This file should be public-key X.509 file or private-key X.509 certificate file.

For example:

ENTPREC((*MBRSET 'pkwareCertAdmin04.cer' () *RQD))

The digital certificate file ‘pkwareCertAdmin04.cer’ will be in the full path of the public certificate store defined in the Enterprise Security Configuration public store(parameter CSPUB). If a passphrase was included, the file would be searched for in the Enterprise Security Configuration private store (parameter CSPRIV).

- If the type is *INLIST: The recipient string defines a full file name of an input list file that contains records of ENTPREC shortcut parameters. The type of file will in the QSYS library file system if TYPLISTFL(*DB) is set and will be a path file name in the IFS if TYPLISTFL(*IFS) is set. The format of the ENTPREC shortcut parameters are define below in the *INLIST usage section.
- If type is *SPONSOR, the recipient string is the sponsor recipient file stored in the ‘…/Sponsor/Recip’ folder. If the recipient string is all numeric, the name will automatically be formatted as R0000000.p7, assuming that the number is the sponsor ID number.

Passphrase (Private Cert Passphrase)

The passphrase is required only if the certificate that is being selected is a private certificate. This option should be omitted if a public certificate will be utilized.

Required (*RQD|*OPT|*SAME)

If *RQD, then this recipient must be found during the selection, and the certificate must be valid, or the ZIP/UNZIP run will fail.

Usage Notes:

The ZIP process only requires a X.509 public-key format certificate to encrypt files. The UNZIP process requires X.509 private-key format certificate file to decrypt files and this will the input of a passphrase.

For inlist that contains a passphrase to open a private certificate, make sure that the security is sufficient to only allow the owner of the certificate to have read access. Otherwise this would leave a security hole where other users could browse the passphrase.

***INLIST Usage:**

If *INLIST is defined on the ENTPREC parameter, then the recipient field will be a file that Smartcrypt will read to include recipients. The format is very similar to the ENTPREC parameter describe above except each line recipient starts with “{RECIPIENT=” and is terminated by the “}” character with the semi-colon “;” as a separator for each entry.

{RECIPIENT=Lookup Type; Recipient; Passphrase; Required}

Lookup Type	See Lookup Type in ENTREC excluding the INLIST

Recipient	See Recipient in ENTREC.
Passphrase	See Passphrase in ENTREC.
Required	See Required in ENTREC, but use RDQ for *RQD and OPT for *OPT.

Sample 1: tstpriv_db4.inlist.

```
{RECIPIENT=DB;EM=PKTESTDB4@nowhere.com;PKWARE;RQD}
```

Sample 2: tstpriv_mb3.inlist.

```
{RECIPIENT=MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

Sample 3: tstpubl1.inlist.

```
{RECIPIENT=MBRSET;pktestdb3.crt;;RQD}
{RECIPIENT=MBRSET;pktestdb4.crt;;OPT}
```

Sample 4: tstpubl2.inlist.

```
{RECIPIENT=DB;EM=PKTESTDB3@nowhere.com;;RQD}
{RECIPIENT=DB;CN=PKWARE Test4;;OPT}
```

ENCRYPOL

Requires Smartcrypt

```
Encryption Filters:
    Validate Level     . . . . .   *SYSTEM       *VALIDATE, *WARN, *NONE...
       Filters . . . . . . . . .   *SYSTEM       *SYSTEM, *ALL, *NONE...
               + for more values
```

Or

ENCRYPOL(*WARN (*SYSTEM))
ENCRYPOL(*WARN (*ALL *NOTTRUSTED))
ENCRYPOL(*SYSTEM (*ALL *NOTEXPIRED))

This parameter defines the processing options and filters that should apply when the ENTPREC is used to encrypt files with certificate keys.

Validate Level (*VALIDATE |*WARN |*SYSTEM)

The validate level specifies the type of encryption certificate error processing that is used if certificates are specified in ENTPREC. If *SYSTEM is specified, the enterprise setting from PKCFSEC is used. If the enterprise setting is defined as lockdown, then this parameter cannot be revised and a warning will be issued if a change is detected.

- *SYSTEM - Indicates the authentication processing that is set in the environmental setting will be used.
- *VALIDATE – Indicates that when encryption with certificates (ENTPREC parm) takes place and a failure based on the filters occurs, the run will be considered a failure and the message issued at the end will indicate one or more errors during the run.
- *WARN - Indicates that when encryption with certificates (ENTPREC parm) takes place and a failure based on the filters occurs, the failure is only considered a warning. The messages at the end of the run will not consider any failed filters for encryption certificates as errors.

Filters (<u>*SYSTEM</u> |*ALL |*NONE |*TRUSTED |*EXPIRED |*REVOKED |*NOTTRUSTED |*NOTEXPIRED |*NOTREVOKED)

The ENTPREC certificate filter policies settings are defined in the enterprise security file supplied by the Smartcrypt administrator (see PKCFGSEC). These global policy settings can be revised with sub-parameter values, but if the enterprise setting is defined as lockdown, this parameter cannot be revised and a warning will be issued if a change is detected. The variables are cumulative from the global setting.

- **<u>*SYSTEM</u>** – All filter policies are from the global settings.
- ***ALL** - This sub-parameter activates all levels of authentication. If followed by negating sub-levels, then all but those negating levels are activated. For example: *ALL, NOTEXPIRED means that expired certificates will not cause an authentication error, but TRUST and REVOKE must both be satisfied.
- ***NONE** – Will negate all the policies.
- ***TRUSTED** – This sub-parameter signifies that the entire certificate authority chain must be validated. This includes locating the root (“self-signed”) certificate on the local system.
- ***EXPIRED** – This sub-parameter signifies that certificate date range validation should be performed on the certificates (including the certificate authority chain). Although the term “expired” is used, a certificate that has not yet reached its valid data range specification will fail.
- ***REVOKED** - A certificate owner may request that the issuing certificate authority declare a certificate to be revoked and thereby no longer consider that certificate to be valid. The encryption certificate request will fail if any of the certificates in the trust chain are found to have been revoked or if the revocation status could not be determined.
- ***NOTTRUSTED** – Negates the *TRUSTED filter.
- ***NOTEXPIRED** - Negates the *EXPIRED filter.
- ***NOTREVOKED** – Negates the *REVOKED filter.

ERROPT

ERROPT(*END | *SKIP | *WARN)

Specifies what action to take if an error occurs while processing (selecting or compressing) a spool file or using the iPSRA.

If iPSRA with the command is SAVCHGOBJ and no files were found to archive, then the error AQZ0368 “Save Command Found Nothing to Do” is issued. AQZ0368 follows the rules of the ERROPT parameter.

If iPSRA is active in the run and an exception occurs (for example, the library was not found with SAVLIB), the message AQZ1023 follows the rules of the ERROPT parameter.

The allowable values are:

***END**	PKZIP will end without completing the compression of the file and the archive will not be built nor updated.
***SKIP**	The program will skip the file with the input error and continue to process all other files to completion. If no other errors occur, the run will issue the AQZ0020 to indicate a successful run. A valid archive is built.
***WARN**	The *WARN option performs the same as *SKIP option but the error message AQZ0022 will be issued at the end to indicate that an error occurred.

EXCLFILE

EXCLFILE(*NONE| *path/filename*)

This parameter specifies the file containing the list of files to be excluded. This can be used with or without the EXCLUDE parameter. See parameter TYPLISTFL for file system type information.

***NONE**	No list file will be processed.
path/filename	Enter the file path and name of the file to process. The layout depends on which file system you want the file created.

Library File System:
The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

EXCLUDE

EXCLUDE(file_specification1, file_specification2,... file_specification *n*)

Specifies the files and file specification patterns that will be excluded from the PKZIP run. One or more names can be specified. Each name should be in the IBM i OS file system format, such as, QSYS is library/file(member) and IFS is directory/file, and can include wildcards "*" and "?."

Note: If TYPE(*VIEW) is being used, then the format for these names is the MS/DOS format.

The PKZIP program can also exclude file specifications by using the list file parameter EXCLFILE with a list of names to exclude.

Please refer to "File Selection and Name Processing" in Chapter 1 for details of file specification formatting.

The valid parameter values for the FILES keyword are as follows:

'*file_specification1*'

'*file_specification2*'

'*file_specification n*'

EXTRAFLD

EXTRAFLD(*YES|*NO)

Specifies if the basic ***Smartcrypt***[i] extended file attributes should be stored in the archive. Some basic file attributes are record size, library text description, file text description, etc.

The allowable values are:

***YES**	Store the basic normal IBM i file attributes. This is the default and will be the same as coding *BOTH.
***NO**	Do not store any extended attributes.
***CENTRAL**	Stores the basic normal IBM i file attributes in only the archive's central directory. This will reduce the overall archive size by only storing the attributes in the Central.
***LOCAL**	Stores the basic normal IBM i file attributes in only the archive's local directory. *Warning:* PKUNZIP only utilizes the central directory for extended data attributes.
***BOTH**	Stores the basic normal IBM i file attributes in both the archive's central directory and local directory.

Migration consideration: if the archive will be processed by an earlier release of PKZIP for iSeries™ and the attributes are required, then *BOTH should be coded.

FILES

FILES(file_specification1, file_specification2,... file_specification *n*)

Specifies the files and file specification patterns that will be selected in the PKZIP process. One or more names can be specified. Each name should be in the IBM i OS file system format, such as, QSYS is library/file(member) and IFS is directory/file, and can include wildcard “*” and “?.” For the IFS, the path and file name can up to 256 characters and can contain embedded spaces.

If the FILES parameter starts with a question mark (?) or a dash (-), then PKZIP assumes that a Save command is being entered to activate the iPSRA feature. For details on how to enter iPSRA commands, see Chapter 6.

The key word “*COPY” as an option of FILES parameter, will copy the files from the input archive to the new archive. This can be used when creating a new archive with a different name and avoid selecting any new files.

Note: If TYPE(*VIEW) or TYPE(*DELETE) is being used, then the format for these names is the MS/DOS format.

The PKZIP program can also have file specifications selections to include by using the list file parameter INCLFILE with a list of names to select.

Files may also be excluded. See the EXCLUDE parameter.

Please refer to “File Selection and Name Processing” in Chapter 1 for details of file specification formatting.

The valid parameter values for the FILES keyword are as follows:

'file_specification1'

'file_specification2'...

'file_specification n'

FACILITY (Algorithm Facilities)

Requires Smartcrypt for IBM i V5R3M0 or above

```
Algorithm Facilities:
    Encryption:        . . . . .   *DFT          *DFT, PKSW, IBMSW...
    Hashing:           . . . . .   *DFT          *DFT, PKSW, IBMSW...
```

Or

FACILITY((IBMSW_PKSW) (*DFT))

FACILITY defines the Encryption and Hashing Algorithm API’s that are available and their sequences. At this time there are only two facilities of APIs 1). PKWARE and 2). IBM Software Security.

Currently there are 2 entries for the FACILITY parameter: Encryption, and Hashing.

Encryption: (*DFT | PKSW | IBMSW | PKSW_IBMSW | IBMSW_PKSW)

Sets the encryption facility API to use in the run.

***DFT**	Use the encryption facility specified in the environment setting defined in the PKCFGSEC parameter FACENC

PKSW	Use PKWARE API for encryption
IBMSW	Use IBM Software API for encryption
PKSW_IBMSW	Both PKWARE API and IBM Software API are available for encryption, but use PKWARE API if available
IBMSW_PKSW	Both IBM Software API and PKWARE API are available for encryption, but use IBM Software API if available

Hashing: (*DFT | PKSW | IBMSW | PKSW_IBMSW | IBMSW_PKSW)

Sets the hashing facility API to use in the run.

***DFT**	Use the hashing facility specified in the environment setting defined in the PKCFGSEC parameter FACHASH.
PKSW	Use PKWARE API for Hashing.
IBMSW	Use IBM Software API for Hashing.
PKSW_IBMSW	Both PKWARE API and IBM Software API are available for Hashing, but use PKWARE API if available.
IBMSW_PKSW	Both IBM Software API and PKWARE API are available for Hashing, but use IBM Software API if available.

FILESTEXT

FILESTEXT(*NO|*ALL|*NEW|*UPDATE)

Specifies if PKZIP allows the editing (and the type of editing performed) of a file's text comments that are stored in an archive.

The allowable values are:

***NO**	No comment editing (the default).
***ALL**	Add comments or edit comments for all files in the archive.
***NEW**	Add comments only for new files that are added to the archive.
***UPDATE**	Add or edit the comments of files that are added, updated, or freshened in the archive. Only file comments of files that are affected by a change are eligible for editing.

FILETYPE

FILETYPE(*BINARY|*DETECT|*EBCDIC|*FIXTEXT|*FIXTXTN|*TEXT)

Specifies whether the files selected are treated as text or binary data. For text files added to an archive, trailing spaces in each line are removed, the text is converted to ASCII (based on the translation tables) by default, and a carriage return and line feed (CR/LF) are added to each line before the data is compressed into the archive. Binary files are not converted.

The default is *DETECT; where PKZIP attempts to make a determination based on the nature of the data itself. The program will read in a portion of the data, evaluate it, and determine the appropriate process.

Note: This will lower performance time. A message will display the type used when compressing.

Use of text file option is usually faster because PKZIP has to process less data than with *BINARY, but more processing may also take place to perform the translation.

If the file is a SAVF or a database file (with DBSERVICE(*YES)), then the file will be processed as binary regardless of what option is specified.

***BINARY**	Specifies that the files selected are binary files and no translation should be performed.
<u>*DETECT</u>	PKZIP will try to determine the data type of text or binary.
***EBCDIC**	Specifies that the files selected are text files and leaves it in EBCDIC without performing any translation. This is good only if the files are to be used on an IBM i or IBM-type mainframe. If they will be unzipped to a PC file, then a translation from EBCDIC to ASCII is required.
***FIXTEXT**	Specifies that the files selected are text files with a fixed record length based on the IBM i file's record length and translation will be performed using the translate tables specified in the TRAN option. This means the compressed file will contain records with trailing spaces followed by a CR and LF. This is only valid for QSYS library file types as files in the IFS do not contain a record length.
***FIXTXTN**	This option is the same as *FIXTEXT except it will accept files with null capable fields, therefore eliminating i/OS data mapping errors.
***TEXT**	Specifies that the files selected are text files and translation will be performed using the translate tables specified in the TRAN option.

<u>FNE</u>

Requires Smartcrypt

FNE(*YES|<u>*NO</u> *YES|<u>*NO</u>)

```
File Name Encryption     :
   Create FNE Archive  . . . . .   *NO           *NO, *YES
   Overwrite In FNE  . . . . . .   *NO           *NO, *YES
```

FNE(*NO *NO)

Specifies the activation and use of the file name encryption feature (see “What is File Name Encryption?” and the System Administrator’s Guide for more information).

The first option controls the creation of an archive with file name encryption.

<u>*NO</u>	Do not create the new/updated archive as a file-name-encrypted archive.
***YES**	Create the new or updated archive as a file-name encrypted archive. If the archive exists, then the security features will be defined by the inputted FNE archive. If no archive with this name exists, the new file-name-encrypted archive will use the encryption method defined with ADVCRYPT parameter and the PASSWORD and/or ENTPREC parameters.

The second option controls the overwriting of a file-name-encrypted input archive to remove the file name encryption. This option is used with the first option of *NO (do not create a file-name-encrypted archive), and with an existing file-name-encrypted archive is input for update. Coding this option to *YES indicates that you know that the input archive is file-name-encrypted and you want to overwrite it to produce an archive that is not file-name-encrypted.

<u>*NO</u>	Do not allow an existing file-name-encrypted archive to be changed to a non-file-name-encrypted archive.
***YES**	Allow an existing file-name-encrypted archive to be changed to a non-file-name-encrypted archive when archive is updated.

<u>FTRAN</u>

FTRAN(<u>*ISO88591</u> |*INTERNAL| *Member Name*)

Specifies the translation table for use in translating file names, comments, and passphrase from the IBM i EBCDIC character set to the character set used in the archive file (normally ASCII character set). A default internal table is predefined. See Appendix D for additional information.

***<u>ISO88591</u>**	The predefined internal table for translation. This table provides translation that is consistent with the ISO 8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.
***INTERNAL**	To provide some compatibility to pre V8 version, *INTERNAL will use the predefined internal tables that were the default in V5 PKZIP.
membername	Specify the member name in the file PKZTABLES that will be parsed and used to translate "file names and comments" files to the archive character set. The member should have the exact format of member

ISO9959_1 in file PKZTABLES. See Appendix D for information on defining translation tables.

GZIP

GZIP(*YES|<u>*NO</u>)

If this option is set to *YES, PKZIP will create a compressed archive in the GZIP format. The GZIP format only allows for one file or member per archive and all text data is stored in ISO 8859- 1 (LATIN- 1) character set. The GZIP format is very different from the ***Smartcrypt****i* archive format, a program that can process ***Smartcrypt****i* archives will not necessarily process a GZIP archive correctly. The GZIP archive created conforms to the GZIP specifications RFC1951 and RFC1952.

Do not use this option if the archive is to be unzipped on another platform where GZIP compatibility is not confirmed.

The allowable values are:

***YES**	PKZIP will create a compressed archive in the GZIP format.
<u>*NO</u>	PKZIP will create an archive in the ***Smartcrypt****i* format. This is the default.

IFSCDEPAGE

IFSCDEPAGE(<u>*NO</u>| *Code-Page*)

If this option is set to *NO, PKZIP will read IFS files using the code page that is registered for the file. Otherwise, PKZIP will read IFS files with the specified code page.

This parameter also controls the spool file ASCII conversion for *TEST and *PDF documents. When *NO is specified for spool Files, the conversion will use code page 819.

The allowable values are:

<u>*NO</u>	PKZIP will read IFS files with the code page registered for the file. If the file is a spool file, the code page 819 will be used. This is the default.
Code-Page	PKZIP will read IFS files with the specified code page value. If the file is a spool file, it is the code page that a spool file will use for ASCII translation.

INCLFILE

INCLFILE(<u>*NONE</u>| *path/filename*)

This parameter specifies the file containing the list of files to be selected for inclusion. This can be used with or without the FILES parameter. See parameter TYPLISTFL for file system type information.

NONE	No Include list file will be processed. This is the default.
path/filename	Enter the file path and name of the file to process. The layout depends on which file system you want to create the file in.

Library File System:
The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

MSGTYPE

```
Outlist Details:
       Type . . . . . . . . . .   *BOTH         *BOTH, *SEND, *PRINT
       License Info.  . . . . .   *NORMAL        *NORMAL, *SHORT, *NONE
```

Or

MSGTYPE(*SEND)
MSGTYPE(*BOTH *SHORT)
MSGTYPE(*BOTH *NONE)
MSGTYPE(*PRINT)

This parameter specifies where displayed output will be outputted and the type of licensing splash screen to provide.

Detail Type (*BOTH |*PRINT |*SEND)

Specifies where the display of messages and information should be shown. The PKZIP program has the ability to send messages that appear on the log and/or the ability to print to stdout and stderr. If working interactively, stdout and stderr will show upon the dynamic screen. If submitted via batch, you can override them to print in an OUTQ or build a CL and save them to an outfile.

***BOTH**	Send the information to the log with send message commands and also to stdout and stderr
***PRINT**	Send the information to stdout and stderr
***SEND**	Send the information to the log with send message commands

License Info (*NORMAL |*SHORT |*NONE |*COPYRIGHT)

Specify what type of ***Smartcrypt***[i] license and copyright information to display.

***NORMAL**	Displays all license and copyright information
***SHORT**	Displays base licensing/copyrights
***NONE**	Displays only registration information
***COPYRIGHT**	Displays Copyright and trademark details from $COPYRIT file in the product distribution library.

NSSRULES

Requires Smartcrypt

```
NSS Process Settings:
    NSS Classify Archive  . . . .   *SYSTEM       *VALIDATE, *WARN, *NONE...
               + for more values
    NSS Check Archive State . . .   *SYSTEM       *NO, *WARN, *FAIL, *SYSTEM
```

Or

NSSRULES(INACTIVE)
NSSRULES (SECRET_SUITEB_REQPLUS *WARN)
NSSRULES (TOPSECRET_SUITEB_STRICT *SYSTEM)

The NSS rules parameter controls the enterprise settings for adhering to their NSS process. There are currently two option settings for NSSRULES.

NSS Classify Archive (<u>*SYSTEM</u> | *NO | INACTIVE | SECRET_SUITEB_REQPLUS | SECRET_SUITEB_STRICT | TOPSECRET_SUITEB_REQPLUS | TOPSECRET_SUITEB_STRICT)

The NSSCLASSIFY setting governs enablement of SECRET and TOP SECRET classification associated with Suite B cryptographic algorithms as specified by the National Institute of Standards and Technology (NIST) for protecting National Security Systems (NSS). Suite B includes cryptographic algorithms for encryption, digital signature, and hashing.

The default is *SYSTEM and, unless it is modified, Smartcrypt will use the enterprise setting from PKCFSEC .

- **<u>*SYSTEM</u>** - All filter policies are from the global settings.
- ***NO / INACTIVE** - No classification criteria enforcement is done.
- **SECRET_SUITEB_ REQPLUS / SS_REQP** - A restriction to algorithms and key strength specifications associated with Classification level SECRET or better are to be enforced.
- **SECRET_SUITEB_STRICT / SS_STRICT** - A restriction to algorithms and key strength specifications associated with Classification level SECRET are to be enforced exactly.
- **TOPSECRET_SUITEB_REQPLUS / TS_REQP** - A restriction to algorithms and key strength specifications associated with Classification level TOP SECRET or better are to be enforced.
- **TOPSECRET_SUITEB_STRICT / TS_STRICT** - A restriction to algorithms and key strength specifications associated with Classification level TOP SECRET are to be enforced exactly.

NSS Check Archive State (*NO | *WARN | *FAIL |*SYSTEM)

The NSSCHECK setting is used during the updating of an archive in concert with a NSSCLASSIFY specification level to be checked for authentication.

The default is *SYSTEM and, unless it is modified, Smartcrypt will use the enterprise setting from PKCFSEC.

- **<u>*SYSTEM</u>** - Indicates the authentication processing that is set in the environmental setting will be used.
- ***NO** - No check will take place.
- ***WARN** - Processing continues. Archive updating is permitted to complete. A warning message AQZ0060 “Smartcrypt ZIP ending with Warnings for <Suite B Issues>” will be returned instead of messages AQZ0020 or AQZ0022.
- ***FAIL** – Archive updating processing will be terminated. For all actions, the message AQZ0022 “Smartcrypt ZIP Completed with Errors” will be returned when a mismatch occurs.

PASSWORD

PASSWORD(Archive Passphrase)

Specifies an encryption passphrase for files being added to an archive. This passphrase may be up to 260 characters in length and is case sensitive. All files selected for archiving will be encrypted using the specified passphrase. If a contingency key is coded for the enterprise, certificate-based encryption is done for the key in addition to the passphrase-based encryption. The length range of the passphrase for Smartcrypt is defined for the enterprise settings by PKCFGSEC.

Note: There is no way to extract the passphrase used from the archive data. If the passphrase is forgotten, the file will become inaccessible. If files in an archive need to have different passphrases, PKZIP must be run for each passphrase required.

Since the passphrase is entered in EBCDIC, the translation table referenced in the FTRAN parameter is used to translate it to an ASCII format. In order to be able to use this passphrase on other platforms, care should be taken to assure that there is a correct translation of the EBCDIC character to a valid ASCII character. For this reason, creating a passphrase using standard character set characters is safer for use in other environments. Care should also be taken when using the FTRAN() override with a passphrase encryption. To extract passphrase-protected files, the same FTRAN() override option is required.

If the contents of the PASSWORD() parameter starts with the key word *INLIST;, then the passphrase will be retrieved from the Inlist file defined in the PASSWORD() parameter.

***INLIST Usage**

To utilize the inlist file for a passphrase, the PASSWORD parameter must start with the keyword *INLIST;. If the inlist file is not from the same file system that is set with the TYPLISTFL(*IFS/*DB) parameter, then code a *DB; or *IFS; to describe the file system where the following inlist will reside. Following the *INLIST; or *IFS;/*DB;, the file name should be specified. Using the inlist method for passphrase allows the opportunities to secure a file of a passphrase with the IBM i object authorities. For more information on INLIST see Appendix C.

Examples of PASSWORD() passphrase inlist file coding:

- **PASSWORD('*INLIST;ATEST/PPINLIST(MBR01)')**

  where TYPLISTFL(*DB) and the file is PPINLIST in library ATEST for file member MBR01

- **PASSWORD('*INLIST;*DB;ATEST/PPINLIST(MBR01)')**

  demonstrates overriding the TYPELISTFL

- **PASSWORD('*INLIST;/myroot/PKINLIST/PP01.inl')**

  where TYPLISTFL(*IFS) and the path to the inlist file is '/myroot/PKINLIST/' with stream file name of 'PP01.inl'

- **PASSWORD('*inlist;*IFS;/myroot/PKINLIST/PP01.inl'))**

  demonstrates overriding the TYPELISTFL

The passphrase inlist file must contain "PASSPHRASE={" and be terminated by the "}" character. The passphrase used will be all of the bytes that exist between the {} not including the { or the }. Null bytes and end-of-record bytes are ignored. Therefore the passphrase structure should be only on one record of an inlist file.

Example of contents of an inlist file:

```
PASSPHRASE={x12345678901234x}
```

The passphrase used will be x12345678901234x in EBCDIC. This will then be translated to ASCII using the FTRAN parameter.

Care should be taken that the file is EBCDIC and has a correct code page. It will use all bytes of data between the {}, even non-displayable bytes.

After a passphrase inlist file is set up, it should be tested with both PKZIP and the PKUNZIP command.

HEXKEY:<*display-hex value*>

This coding form (HEXKEY: in mixed case) provides for the specification of a symmetric cipher ZIP archive file protection.

HEXKEY: is used to provide a binary key for qualifying algorithms to protect files in a ZIP archive.

The key is provided in a display-hex format, where 2 hexadecimal character values comprise 1 byte (8 bits) of key information. The number of hex characters specified must match the selected encryption method or ADVCRYPT key strength for the cipher to produce the correct strength.

Encryption Method	*Required Hex Characters*
3DES	42 characters (168 bits)
RC4	32 characters (128 bits)
AES128	32 characters
AES192	48 characters
AES256	64 characters

**Example: For AES128; PASSWORD('HEXKEY:12345678123456781234567812345678')
VPASSWORD('HEXKEY:12345678123456781234567812345678')**

PKOVRTAPF

```
New Archive Tape Overrides:
    Tape Device        . . . . .   *TAPF          Tape Device
    Tape File Label    . . . . .   *TAPF          Tape Header
    Tape Sequence Nbr  . . . . .   *TAPF          1-16777216, *TAPF, *END
    File expiration date . . . .   *TAPF          Date, *NONE, PERM, *TAPF
    End Of Tape Option . . . . .   *TAPF          *TAPF, *REWIND, *UNLOAD...
    Shadow Dir File    . . . . .   *CSDF          *CSDF, *NO
```

Or

**PKOVRTAPF(*TAPF 'ARCHIVE_TEST01' 1 '11/08/2005' *TAPF)
PKOVRTAPF(*TAPF 'ARCHIVE_TEST02' *END *TAPF *LEAVE, *NO)
PKOVRTAPF(TAP02 'ARCHIVE_TEST03' *END *PERM *LEAVE *CSDF)**

This parameter defines the override options for the tape device file specified in the archive parameter. These options are active only if TYPARCHFL(*TAP) is set to write the archive directly to a tape. When *TAPF is specified, the value from the current tape device file is used. For more information on these options, refer to the CRTTAPF command.

Tape Device (Tape Device |*TAPF)

Overrides the DEV() parameter of the tape device file. Specifies the name of a tape device used with this device file to perform output data operations.

Tape File Label (label string |*TAPF)

Overrides the LABEL() parameter of the tape device file. Specifies the data file identifier of the data file that is being processed by this tape device file. The data file identifier is defined for standard-labeled tapes and is stored in the header label immediately before the data file that the header describes. For more details on the specification for this 17-byte option, refer to the CRTTAPF command and tape label processing.

Tape Sequence Number (*END |sequence nbr |*TAPF)

Overrides the SEQNBR() parameter of the tape device file.

This specifies the file sequence number of the data file on the tape being processed. For standard-labeled tapes, this four-position file sequence number is read from the first header label of the data file.

- ***END** - The file is written after the last file currently on the tape. If no files exist on the tape then it will be fist file on the tape.
- **Sequence number** - Specifies the file sequence number of the file being processed on this tape.

Tape File Expiration Date (Date |*NONE |*PERM |<u>*TAPF</u>)

Overrides the EXPDATE() parameter of the tape device file. This option specifies the expiration date of the data file used by this device file. The data file is protected and cannot be written over until the specified expiration date has past.

- ***NONE** - No expiration date for the data file is specified. The file is not protected.
- ***PERM** - The data file is protected permanently. The date written on the tape is 999999.
- **Date** - Specify the date on which, and beyond which, the data file is no longer protected. The date format should be mm/dd/yyyy such as ‘11/08/2016’.

End of Tape Option (<u>*TAPF</u> |*REWIND |*UNLOAD |*LEAVE)

Overrides the ENDOPT() parameter of the tape device file. This option specifies the operation that is automatically performed on the tape volume after the operation ends. If more than one volume is included, this parameter applies only to the last tape volume used; all other tape volumes are rewound and unloaded when the end of the tape is reached.

- ***REWIND** - The tape is rewound, but not unloaded.
- ***UNLOAD** – The tape is automatically rewound and unloaded after the operation ends.
- ***LEAVE** – The tape does not rewind or unload after the operation ends. It remains at the current position in the tape drive.

Shadow Dir File (<u>*CSDF</u> |*NO) *New Option*

Build a shadow directory file for efficiency when viewing or extracting a tape archive.

- ***CSDF** - The default is to build the Shadow Directory File after building the tape archive file.
- ***NO** – Do not create Shadow Directory File. Only the tape archive file will be written to tape.

<u>SELFXTRACT</u>

SELFXTRACT (<u>*MAINTAIN</u>| *REMOVE | Self extraction member name)

This licensed feature specifies the action to take concerning self-extracting archives. The actions are to maintain the current archive as is, create the new archive with a self-extracting preamble, or to remove the self-extracting preamble if one exists in the archive.

The self-extracting programs are held as binary entities in the ***Smartcrypt***^{i} library in the file PKZIPSFX. The appropriate member is loaded and the executable data copied to the beginning of the archive as a preamble when requested.

The resulting archive can still be processed by ***Smartcrypt***^{i} as a normal ZIP archive.

The allowable values are:

<u>MAINTAIN</u>	If the current archive contains a self-extracting preamble, it will be maintained at the beginning of the updated archive.
Self extraction member name	See active table below.
***REMOVE**	If the current archive contains a self-extracting preamble, it will be removed.

Self-extraction member name active table:

self_extraction_program_name ***Values***	***Description***
SF6AIX	IBM AIX Version 4.3 and above
	Size: 667K
	AIX is an alias for this value.
SF6HP	HP/UX Version 10.20 and above
	Size: 774K
	HP_UNIX is an alias for this value.
SF6LNX2I	LINUX running the 2.4 or later kernel on x86 with glibc-2.2.4
	Size: 410K
	LINUXINTEL is an alias for this value.
SF6SUN	Sun Solaris 2.6 and above
	Size: 431K
	SUN_UNIX is an alias for this value.
SFAWINC	Microsoft Windows command line (Windows 98, Windows Me, Windows NT 4.0, Windows 2000, and Windows XP systems. NT 4.0, must include Microsoft Internet Explorer 4.0 or greater)
	Size: 353K
	WINDOWS is an alias for this value.
SFAWING	Microsoft Windows GUI (same as SFAWINC)
	Size: 357K
SFCAIX	IBM AIX Version 5.1 and above
	Size: 2,354K
SFCHP	HP/UX Version 11.00 and above
	Size: 2,496K
SFCLNX2I	LINUX running the 2.4 or later kernel on x86 with glibc-2.2.5
	Size: 1,523K

self_extraction_program_name *Values*	***Description***
SFCSUN	Sun Solaris 2.8 and above Size: 1,940K
SFCWINC	Microsoft Windows command line (Windows 2000, Windows 2003, Windows XP Pro and Windows Vista systems) Size: 736K
SFCWING	Microsoft Windows GUI (same as SFCWINC) Size: 796K
SF2AIX	IBM AIX Version 4.0 and above Size: 148K
SF2HP	HP/UX Version 9.0 and above Size: 145K
SF2LNX2I	LINUX Kernel 2.x for Intel (target system run-time requirements: Reference PKZIP Support Notice #13 02/16/2001 regarding LINUX target system support files ld.so-1.9.5-13.i386.rpm and libc-5.3.12-31.i386.rpm) Size: 93K
SF2SUN	Sun Solaris 2.3 (SunOS 53) and above Size: 119K
SF2WINC	Microsoft Windows (95 and above) Size: 89K

Usage Notes:

The resulting archive can still be processed by ***Smartcrypt***[i] as a normal ZIP archive.

In most cases, to include the path, use STOREPATH(*REL) or STOREPATH(*NOROOT). Do not use STOREPATH(*YES) with self-extracting archives.

Use the ISRTPATH parameter to preset a path to store the files.

When transferring a self-extracting archive to a target system, be sure to transfer the archive in BINARY format and adhere to requirements for executables in that environment. (For example, a Windows program should be saved with an application extension of EXE, and a UNIX file attribute should have executable authorization set via the UNIX *chmod* command).

Four version levels of self-extraction programs are provided for the Windows platform (version 2.5, version 6.1, version 10.0 and version 12) while the UNIX platforms are provided with three self-extraction levels (version 2.5, version 6.1 and version 12). The version 2.5 components are prefixed with “SF2”. The version 6.1 components are prefixed with “SF6”. The version 10.0 components are prefixed with an “SFA” (A being 10 hexadecimal). Finally, the version 12.0 components are prefixed with an “SFC” (C being 12 hexadecimal). In addition to the command line versions of the Windows extractors, GUI versions are also provided.

The self-extraction programs provided at the 2.5 level of PKUNZIP have the following restrictions. Take care to create the self-extracting archive in keeping with these restrictions. If any of the disallowed capabilities are required on the target system, use the 6.1 or higher versions of the self-extractor instead. Archives created using a version 2.5 self-extraction program can be processed by a higher level version of PKUNZIP.

- The number of files in the archive must be limited to 65,535 or less
- Strong encryption is not supported
- Authentication of digital signatures is not supported (although the signatures within the archive will be maintained and can be authenticated by appropriate Smartcrypt products)
- The size of the archive should not exceed 2 gigabytes
- The uncompressed size of individual files should be less than 2 gigabytes (4 gigabytes on some UNIX systems)

Key Functional Differences Among the Self-Extractor Levels

Extractor	*Code Level*	*ZIP64*	*Strong File-data Decryption*	*Strong File name Decryption*	*ZDW Support*
SF2AIX	2.5	No	No	No	No
SF2HP	2.5	No	No	No	No
SF2LNX2I	2.5	No	No	No	No
SF2SUN	2.5	No	No	No	No
SF2WIN	2.5	No	No	No	No
SF6AIX	6.1	Yes	Yes	No	No
SF6HP	6.1	Yes	Yes	No	No
SF6LNX2I	6.1	Yes	Yes	No	No
SF6SUN	6.1	Yes	Yes	No	No
SFAWINC	10.0	Yes	Yes	Yes	No
SFAWING	10.0	Yes	Yes	Yes	No
SFCAIX	12	Yes	Yes	Yes	Yes
SFCHP	12	Yes	Yes	Yes	Yes
SFCLNX2I	12	Yes	Yes	Yes	Yes
SFCSUN	12	Yes	Yes	Yes	Yes
SFCWINC	12	Yes	Yes	Yes	Yes
SFCWING	12	Yes	Yes	Yes	No

SFUSER

SFUSER (*CURRENT|*ALL |User Name List)

Specifies the user names that created spool files that will be selected. This value is ignored if SFJOBNAM is coded.

The allowable values are:

<u>***CURRENT**</u>	Only files created by the user running this command are selected.
***ALL**	Files created by all users are selected.
User Name	Specify up to 10 user names. Only files created by those users are selected.

<u>SFQUEUE</u>

SFQUEUE (<u>*ALL</u> |Name|*LIBS)

Specifies the output queue that will be searched for the spool file selections. If no OUTQ library is specified, it will default to *LIBL.

The allowable values are:

<u>***ALL**</u>	Files on any device-created or user-created output queue are selected.
OUTQ	The OUTQ that will be searched.
OUTQ Library	The library where the OUTQ resides. Defaults to *LIBL.
***LIBS**	*LIBS will search all OUTQ that exist in the specified OUTQ Library. If *LIBS is selected then the library cannot be blank, nor contain *LIBL nor *CURRENT.

<u>SFFORM</u>

SFFORM (<u>*ALL</u> | *STD| Form Type)

Specifies the spool file form type that is on the spool files that will be selected.

The allowable values are:

<u>***ALL**</u>	Files for all form types are selected.
***STD**	Only files that specify the standard form type are selected.
Form Type	Only spool files with this specific form type will be selected.

<u>SFUSRDTA</u>

SFUSRDTA (<u>*ALL</u>| User Data)

The user data tag associated with the spool file to select.

The allowable values are:

<u>***ALL**</u>	Files with any user data tag specified are selected.

User Data	Only spool files with this specific user data tag will be selected.

SFSTATUS

SFSTATUS (<u>*ALL</u> |*READY|*HELD|*CLOSED|*SAVED|*PENDING|*DEFERRED)

Specifies the status of the spool files to be selected. Up to four status definitions can be selected for one run.

The allowable values are:

<u>*ALL</u>	All spool file status will be considered for selection.
***READY**	Only spool files with a status of *READY will be selected.
***HELD**	Only spool files with a status of *HELD will be selected.
***CLOSED**	Only spool files with a status of *CLOSED will be selected.
***SAVED**	Only spool files with a status of *SAVED will be selected.
***PENDING**	Only spool files with a status of *PENDING will be selected.
***DEFERRED**	Only spool files with a status of *DEFERRED will be selected.

SFJOBNAM

SFJOBNAM(<u>Blank|*|</u>Spool File Jobname/User/Job Number)

Specifies the job name, user name, and job number that will be used to select spool files. If anything other than blanks is in SFJOBNAM parameter, it will be used as the primary selection criteria. If any of the three fields (job name, user name, and job number) are specified, then all three fields must be entered and be valid.

The allowable values are:

<u>Blank</u>	This is the default selection. This will cause all other selection criteria to be used for spool files.
*****	The * will cause the current job-name/user-name/job number to be used to select spool files.
Job-name	Specify the name of the job to be selected. If no job qualifier is given, all of the jobs currently in the system are searched for the simple job name.
User-Name	Specify the name that identifies the user profile under which the job is run.
Job-Number	Specify the job number assigned by the system.

SFTARGET

SFTARGET (*SPLF|*TEXT|*PDF|*TEXT1|*TEXT2|*TEXTFC|*PDF|*PDFLETTER|*PDFLEGAL)

Specifies the format of the file that will be stored in the archive.

The allowable values are:

***SPLF**	This is the default selection. This will compress the spool file in a spool file format with all of the spool file attributes. This format is only valid on an AS/400. If the archive is extracted, it will take on the latest spool file settings, such as, job name, user, job number, spool file number, etc. The suffix for this selection is SPLF. Parameter SFTGFILE is required to be *GEN1 for SFTARGET(*SPLF).
***TEXT**	The spool file will be saved in the archived as an ASCII text document. The suffix for this selection is .TXT. Each new page will have a form feed control character.
***TEXT1**	The spool file will be saved in the archived as an ASCII text document. The suffix for this selection is .TXT. Each new page will have a carriage control and line feed control characters.
***TEXT2**	The spool file will be saved in the archived as an ASCII text document. The suffix for this selection is .TXT. Each page will have a carriage control and line feed control characters for blanks lines to fill out a page with the number lines required by the spool file attribute.
***TEXTFC**	The spool file will be saved in the archive as an ASCII Text document. The suffix for this selection is .TXT. Each new page will have a form feed control character.
***PDF**	The spool file will be saved in the archived as a PDF text document. The suffix for this selection is .PDF. The size will be adjusted based upon the width and length of the spool file.
***PDFLETTER**	The spool file will be saved in the archived as a PDF text document. The suffix for this selection is .PDF. The size will be adjusted based upon the width and length of the spool file.
***PDFLEGAL**	The spool file will be saved in the archived as a PDF text document. The suffix for this selection is .PDF. The size will be adjusted based upon the width and length of the spool file.

SFTGFILE

SFTGFILE (|*GEN1|*GEN2|*GEN1P|File Name)

Specifies the how the file name will be stored in the archive.

The allowable values are:

<u>*GEN1</u>	GEN1 is the default selection. This generates a very specific name using most of the spool file name attributes to form the file name so that it will not be duplicated. *GEN1 is required if SFTRAGET is *SPLF. The name will be built as follows: "Job-Name**/**User-Name**/ #**Job-Number**/**Spool-File-Name/Fspool-File-Number.Suffix" "MYJOB/BILLS#152681/INVOICE/F0021.SPLF" The suffix is dependent on the SFTARGET setting.
***GEN1P**	*****GEN1P generates the same file name as *GEN1 except instead of a '/' separator, *GEN1P will use a '.' as name separator.
***GEN2**	*****GEN2 uses the spool file name and appends the spool file number followed by the suffix that is depended on the SFTARGET setting. Caution should be taken in that a duplicate file name in the archive could be created. An example of GEN2 is a spool file INVOICE with spool file number of 21 that will be converted to a text file will generate a file name of INVOICE21.TXT.
File Name	This parameter should only be used when selecting one specific spool file where you want a specific file name.

<u>SPLFILE</u>

SPLFILE (<u>*ALL</u>| Spool File Name)

Specifies the spool file name that will be selected. This parameter is used along with all the other spool file selection parameters to determine the spool files to select.

The allowable values are:

<u>*ALL</u>	This is the default setting. *ALL indicates that spool file name is not important.
Spool File Name	A specific spool file name that will be searched for and selected.

<u>SPLNBR</u>

SPLFILE (<u>*ALL</u>|*LAST| Spool File Number)

Specifies the number of the spool file from the job whose data records are to be selected. If *ALL is coded then all file numbers are considered. This parameter is <u>only valid</u> when the SFJOBNAM parameter or SPLFILE is used. This parameter is used along with all the other spool file selection parameters to determine the spool files to select.

The allowable values are:

***ALL**	This is the default setting. *ALL indicates that spool file number is not important.
***LAST**	The spooled file with the highest number is used.
Spool File Number	A number 1-9999 to specify the number of the spooled file whose data records are to be selected.

SIGNERS

Requires Smartcrypt

```
Signing Certificates        :
    File/Archive  . . . . . . .   *FILE         *FILE, *ARCHIVE, *ALL
    LookUp Type  . . . . . . . .   *DB          *DB, *FILE, *MBRSET, *INLIST
    Signer . . . . . . . . . . .   ______________________________________________
    Passphrase (If Private)  . . . __________________________________________
    Required . . . . . . . . . .   *RQD         *RQD, *OPT
```

Or

SIGNERS((*FILE *MBRSET
'pkwareCertAdmin04.pfx' (passphrase) *RQD))
SIGNERS((*FILE *FILE
'/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.pfx' (passphrase) *RQD))
SIGNERS((*FILE *FILE
'/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.pfx' (‘mypassphrase’) *RQD))
SIGNERS((*FILE *DB
‘EM=bill.somebody@pkware.com' (passphrase) *RQD))
SIGNERS((*FILE *INLIST 'ATEST/INLIST(ENGNEER1)' *N))

This parameter identifies the public key certificate with private key that is to be used to digitally sign files to be added to the archive and/or the archive directory. Multiple signing certificates may be applied to the files but only one signer is allowed to sign the archive directory. Signing an archive by signing its central directory enables people who receive the archive to confirm that the archive as a whole is not changed. By contrast, signing only individual files in an archive enables people to confirm that the particular signed files are unchanged but leaves open the possibility that the archive has had files added or removed.

There are five options for SIGNERS.

Signing Type File/Archive (*FILE |*ARCHIVE |*ALL)

The File/Archive selection determines whether the files, archive or both are to be signed during the ZIP run. Only one signer can be specified for an archive. If the lookup type is *INLIST, then this option will be ignored and will pick up from the records in the inlist file.

- *FILE – All new files being compressed in the run will be signed by this private key and a signature entry will be added to the archive.
- *ARCHIVE – The archive directory will be signed by this private key and a signature entry will be added to the archive.
- *ALL – Both *FILE and *ARCHIVE for the signer will be used.

Lookup Type (*DB |*FILE |*MBRSET |*INLIST)

The lookup type would be the type of signer search that will be used for the signer string to lookup the private key.

- *DB - The signer string is defined to search using the Certificate Locator Database to access the digital certificate and private key.
- *FILE - The signer string is defined to read a specific file in a specific path in the IFS in order to access the digital certificate and private key.
- *MBRSET - The signer string is defined to read this specific file from the enterprise private certificate store to access the digital certificate and private key.
- *INLIST- The signer string defines a specific file that will contain one to many signers. The TYPLISTFL parameter must specify the file type for the inlist.

Signer (The signer string name)

The signer string format depends on what was specified for the lookup type.

- If lookup type is *DB, the signer string will either be an email address or the common name of the certificate. This depends on the configuration setting in PKCFGSEC parameter CERTDB. To override the default selection mode, you can prefix the string with EM= for email, or CN= for the common name.

For example:

```
SIGNERS((*FILE *DB    'bill.somebody@pkware.com'  (passphrase) *RQD)
      (*ARCHIVE *DB    'CN=bill somebody'  (passphrase) *RQD)
      (FILE *DB    'EM=bill.somebody@pkware.com'  (passphrase) *OPT))
```

- If lookup type is *FILE, the signer string is defined to read a specific file in a specific path of the IFS. This file should be public key X.509 with the private key X.509 certificate file.

For example:

```
SIGNERS((*ARCHIVE *FILE '/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.pfx'
(passphrase) *RQD))
```

The digital certificate file with the private key 'pkwareCertAdmin04.pfx' will be in the full path '/yourpath/PKWARE/Cstores/private'.

- If lookup type is *MBRSET, the signer string is defined to read a specific file from the public certificate store and/or the private certificate store of the IFS. This file should be public key X.509 with the private key X.509 certificate file.

For example:

```
SIGNERS((*ALL *MBRSET 'pkwareCertAdmin04.pfx'  (passphrase) *RQD))
```

The digital certificate file 'pkwareCertAdmin04.pfx' will be in the full path of the private certificate store defined in the enterprise security configuration private store (parameter CSPRV). Since the passphrase was included, the file will be searched for in the enterprise security configuration private store (parameter CSPRIV).

- If lookup type is *INLIST the signer string defines a full file name of an input list file that contains records of SIGNERS shortcut parameters. The type of file will exist in the QSYS library file system if TYPLISTFL(*DB) is set and will be a path file name in the IFS if TYPLISTFL(*IFS) is set. The format of the SIGNERS shortcut parameters are defined below in the *INLIST usage section.

Passphrase

This designates the passphrase that is *required* for a private key (PKCS#12 file). When a value is specified, the target must be an X.509 PKCS#12 private key certificate.

The PASSWORD value may contain blanks and is delimited by the closing right parenthesis ")" of the signing command.

Required **(*RQD|*OPT|<u>*SAME)</u>**

If *RQD then this signer MUST be found during the selection and the certificate MUST be a valid certificate with a private key or the run will fail.

Usage Notes:

A NULL file (binary file having zero bytes of data) will be signed. However, note that the digital signature is based on a fixed hash value.

The entire data stream of each file is run through the hash algorithm before compression or encryption. However, file text data is translated before hashing so that the receiving system is able to hash the identical stream after decryption/decompression.

The processor requirement for a file signature is directly related to the size of the file(s) being signed and/or authenticated (see SIGN_HASHALG). Therefore, when processing costs are a consideration, the decision whether to use SIGNERS to sign large files should be based on the business case. Sometimes signers for the archive may be more appropriate. (The directory size is proportional to the number of files in the archive, not the physical size of the file data.)

A separate signing operation is performed for each supplied certificate, for each file. Processor and elapsed time will be impacted in proportion to the number of signatories and files selected.

The number of file signatures that can be held for each file is constrained by a number of factors. These include EXTRAFLD(*YES) and DBSERVICE(*NO), the size of the signatures generated (based on the size of the certificate information), the number of certificates in the authenticating certificate authority chain, the number of different certificate authorities used in association with the signing certificates, if FNE(*YES) is specified, and the number of recipients for certificate-based encryption of files. For planning purposes, typical ZIP operations will support up to 10 file signatories as a rule, although more or fewer may be achieved in practice.

It is important that the passphrase is entered in the correct case. Any variation in case or misspelling will result in a public key certificate access attempt (which will fail for a private key PKCS#12 certificate). Please note that passphrases will be masked out in all output displays.

A local certificate store configuration is required to complete the processing of this command. Even when a direct FILE specification is made to locate the private key certificate, the CS and ROOT certificate store components must be accessible to complete the certificate signing chain within the archive. This information is required to complete authentication processing on the target system when the local certificate store on that system does not contain the certificate authority chain required to validate TRUST (see PKCFGSEC).

Processing will be terminated if none of the requested certificates can be accessed, regardless of the “R” required flag. If multiple requests are made and at least one signature is found, processing will continue normally.

Signed files are tolerated by prior releases of ***Smartcrypt***[i] but are not processed for authentication.

For inlist that contains a passphrase to open a private certificate, make sure that the security is sufficient to only allow the owner of the certificate to have read access. Otherwise this would leave a security hole where others could browse the passphrase.

***INLIST Usage:**

If *INLIST is defined on the SIGNERS parameter, then the signer filed will be a file that Smartcrypt will read to include the signer. The format is very similar to the SIGNERS parameter described above except each line signer starts with “{SIGNERS=” and is terminated by the “}” character with the semi-colon “;” as a separator for each entry.

{SIGNERS=Signing Type, Lookup Type; Signer; Passphrase; Required}

Signing Type	See Signing Type in SIGNERS
Lookup Type	See Lookup Type in SIGNERS excluding the INLIST
Signer	See Signer in SIGNERS.
Passphrase	See Passphrase in SIGNERS.
Required	See Required in SIGNERS, but use RDQ for *RQD and OPT for *OPT.

Sample 1: tstsign_db1.inlist.

```
{SIGNERS=File;DB;EM=PKTESTDB4@nowhere.com;PKWARE;RQD}
```

Sample 2: tstsign_mb2.inlist.

```
{SIGNERS=ARCHIVE;MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

SIGNPOL

Requires Smartcrypt

```
Signing Filters:
    Validate Level     . . . . .   *SYSTEM        *VALIDATE, *WARN, *NONE...
       Filters . . . . . . . . .   *SYSTEM        *SYSTEM, *ALL, *NONE...
               + for more values
       Signing Hash . . . . . . . . *SYSTEM        *SYSTEM, *SHA1, *MD5...
```

Or

SIGNPOL(*WARN (*SYSTEM))
SIGNPOL(*WARN (*ALL *NOTTRUSTED *SYSTEM))
SIGNPOL(*SYSTEM (*ALL *NOTEXPIRED *SHA256))

Signing Hash (*SHA1 | *MD5 | *SHA256 | *SHA384 | *SHA512 |<u>*SYSTEM)</u>

This parameter defines the Hashing algorithm that will be used when creating the digital signature.

- **<u>*SYSTEM</u>** - Indicates the signing hash method for the signing process set in the environmental setting will be used.
- ***SHA1** - The default algorithm generates a 20-byte hash value. This algorithm is supported by all Smartcrypt products.
- ***MD5** - This algorithm generates a 16-byte hash value. It is included for compatibility with older releases of PKZIP on other platforms, which previously supported this algorithm.
- ***SHA256** - A variant of the SHA-2 class of Secured Hash algorithms producing 256 bits (32 bytes) of information. Reference FIPS 180-2.
- ***SHA384** - A variant of the SHA-2 class of Secured Hash algorithms producing 384 bits (48 bytes) of information. Reference FIPS 180-2.
- ***SHA512** - A variant of the SHA-2 class of Secured Hash algorithms producing 512 bits (64 bytes) of information. Reference FIPS 180-2.

For more information on the hashing algorithms and FIPS 180 see SIGNPOL parameters in the System Administrator's Guide.

Validate Level (*VALIDATE |*WARN |<u>*SYSTEM)</u>

The validate level specifies the type of signing processing that should take place if the signer requests encounter an error. If *SYSTEM is specified, the enterprise setting from PKCFSEC is used. If the enterprise setting is defined as Lockdown, then this parameter cannot be revised and a warning will be issued if a change is detected.

- <u>*SYSTEM</u> - Indicates the authentication processing that is set in the environmental setting will be used.

- *VALIDATE - Indicates that when authentication takes place and a failure occurs based on the filters, the run will be considered a failure, and the message issued at the end will indicate one or more errors during the run.
- *WARN - Indicates that when authentication takes place and a failure occurs, the failure is only considered a warning. The messages at the end of the run will not consider any failed filters for signer certificates as errors.

Filters (<u>*SYSTEM</u> |*ALL |*NONE |*TRUSTED |*EXPIRED |*REVOKED |*NOTTRUSTED |*NOTEXPIRED |*NOTREVOKED)

The signing filter policies settings are defined in the enterprise security file supplied by the Smartcrypt administrator (see PKCFGSEC). These global policy settings can be revised with sub-parameter values, but if the enterprise setting is defined as lockdown, this parameter cannot be revised and a warning will be issued if a change is detected. The variables are cumulative from the global setting. The setting of these filters defines what certificates are acceptable for signing.

- **<u>*SYSTEM</u>** - All filter policies are from the global settings.
- ***ALL** - This sub-parameter activates all levels of authentication. If followed by negating sub-levels, then all but those negating levels are activated. For example: *ALL, NOTEXPIRED means that expired certificates will not cause an authentication error, but TRUST and REVOKE must both be satisfied.
- ***NONE** - Will negate all the policies.
- ***TRUSTED** - Each end-entity certificate used in the signature must be traced back to a trusted root certificate. The CACA and CSROOT stores on the local system performing the authentication check will be accessed to determine if the entire certificate chain can be trusted. Although the root (“self-signed”) certificate may be included within the archive, it MUST also exist in the CSROOT store to complete the TRUSTED state.
- ***EXPIRED** - The digital certificates used to originally perform the signing operation contain internal date ranges of validity. The signer operation will fail if any of the certificates in the trust chain are not found to be within their stated data range. Note that an end-entity certificate may have expired at the time that the archive is being accessed, and NOTEXPIRED may be used to continue processing.
- ***REVOKED** - A certificate owner may request that the issuing certificate authority declare a certificate to be revoked and thereby no longer consider that certificate to be valid. The signer operation will fail if any of the certificates in the trust chain are found to have been revoked or if the revocation status could not be determined.
- ***NOTTRUSTED** - Negates the *TRUSTED filter.
- ***NOTEXPIRED** - Negates the *EXPIRED filter.
- ***NOTREVOKED** - Negates the *REVOKED filter.

STOREPATH

STOREPATH(<u>*REL</u> | *NOROOT | *NO |*YES)

Specifies whether to store the full path and file name in the archive, or to just save the file name. If the file is an IFS file type, the path is all directories, from the current directory, to the directory of the file. In the library system, the path is the library and the file name. The member name is considered to be the file name.

The allowable values are:

<u>*REL</u>	Only the RELative path and file name will be stored (i.e., No leading '/'). If an IFS absolute path was used in the file selection, the leading '/' would be removed from the file name in the archive. For example if the file is "/pkzshare/mypath1/mypath2/myfile", then it will be stored as "pkzshare/mypath1/mypath2/myfile".
***NOROOT**	The first node of the path will not be stored. If file type is *DB then the library will not be stored. If the file type is *IFS the first node of the path (not including a starting / if present) will not be stored. For example if the file is "/pkzshare/mypath1/mypath2/myfile", then it will be stored as "mypath1/mypath2/myfile".
***NO**	Store only the file name in the PKZIP archive.
***YES**	Store all paths and the file name in the PKZIP archive including a leading '/' for IFS files using absolute path selection. This option is not recommended and is not valid according to the APPNOTE standards for file names in the archive.

TMPPATH

TMPPATH(<u>*CURRENT</u>| ***pathname*)**

Specifies a directory or library/file in which to build the temporary archive file. While PKZIP is compressing data into an archive, a temporary archive file name is used. The temporary file name is a 10-character name with a prefix of "PZ" followed by a time stamp (PZtttttttt). If this option is *CURRENT, the temporary file is built in the same directory (for library file systems it is same library/file with temporary member) in which the new archive will be stored and is then renamed at the end of the run to the archive name. If an override path is specified, the temporary archive file is built into that specified path, and the file is then copied to its final archive path at the end of the run. The temporary file name and path type will be the same as specified for ARCHIVE. See parameter TYPARCHFL for file system type information. Special libraries (such as QTEMP) are used frequently.

<u>*CURRENT</u>	Specifies that the current archive path will be used (see ARCHIVE) to build the temporary archive file PZxxxxxxxx.

pathname Specifies a path name (if using IFS such as /PKZIP/tempdir) or a library/file (if using the library system).

NOTE 1: When using the QSYS library file system and specifying "qtemp" as the TMPPATH, a dynamic file name and member name is created in the library qtemp. At the end of the run, the file and member are removed. If any other combination of names is used, then a dynamic member name is created and only the member is removed.

NOTE 2: When using the QSYS library file system and specifying a TMPPATH, there may be a slight performance degradation because the archive file will have to be copied from one library/file to another library/file. Otherwise, if *CURRENT is used, the file member name will only be renamed.

TRAN

TRAN(<u>*ISO88591</u> |*INTERNAL| *Member Name*)

Specifies the translation table for use with translating "data" from the IBM i EBCDIC character set to the character set used in the archive file (normally the ASCII character set). A default internal table is predefined (see Appendix D).

<u>*ISO88591</u>	The predefined internal table for translation. This table provides translation that is consistent with the ISO 8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.
***INTERNAL**	To provide some compatibility to pre V8 version, *INTERNAL will use the predefined internal tables that were the default in V5 PKZIP.
Member Name	Specifies the member name in the file PKZTABLES that will be parsed and used to translate data files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES (see Appendix D for information on defining translation tables).

TYPARCHFL

```
Archive File:
       Type . . . . . . . . . . .   *DB           *DB, *IFS, *TAP, *XDB
       Check ZIP64  . . . . . . .   *NONE         *NONE, *WARN, *FAIL
```

Or

TYPARCHFL (*IFS)
TYPARCHFL (*DB *WARN)
TYPARCHFL (*IFS *FAIL)

TYPARCHFL (*TAP)

This parameter specifies the file system to create the archive and the archive constraints.

Archive Type (<u>*DB</u> |*IFS |*TAP |*XDB)

Specifies the type of file system in which the archive file will exist (see parameters ARCHIVE and TMPPATH for additional information).

<u>*DB</u>	Archive files are to be in the QSYS library file system. Even though *DB is working with archive files that are in the QSYS library file system, the IFS is utilized for performance and for large file support (ZIP64). To provide an option for archive file creation utilizing exclusively the QSYS library system, use TYPARCHFL(*XDB), which supports OS400 features such as Adopt Authority.
***IFS**	Archive files are to be in the integrated file system (IFS).
***TAP**	Archive file will be written directly to tape. The ARCHIVE parameter MUST be a tape device file or have attribute *TAPF. The tape device file PKTAPEO1 is distributed with PKZIP.
***XDB**	The archive files are to be in the QSYS library file system and will exclusively use the QSYS library file system during processing. This will force the Check ZIP64(*FAIL). This option will not support Large File Support or ZIP64.

Check ZIP64 (*NONE |*WARN |*FAIL)

Specify the severity of message and return code when creating or updating an archive and ZIP64 processing is required.

<u>*NONE</u>	No action or message when ZIP64 constraint exceeded.
***WARN**	Warning message AQZ0613 will be issued but processing will continue.
***FAIL**	Failure message AQZ0614 will be issued and process will cease without building a new archive.

This feature may be of value when creating archives intended for distribution to systems that may not be able to handle the ZIP64 processing attributes. This may be due to the UNZIP software being used on the target system or the file system for the related OS. (For example, some UNIX or Windows FAT file systems cannot handle file sizes greater than 4 gigabytes).

Triggers for this option include:

- More than 65,535 files are being placed into the archive
- One or more source files are greater than 4 gigabytes in size
- The amount of data written to the archive exceeds 4 gigabytes

TYPE

TYPE(*ADD|*DELETE |*FRESHEN|*MOVEA|*MOVEF|*MOVEU|*UPDATE)

The TYPE keyword specifies the type of action PKZIP should perform on the ZIP archive.

The possible actions are:

***ADD**	The *ADD option is the default and adds a selection of files to the archive file. If an archive is already present, it will be written over by the new archive file.
***UPDATE**	The *UPDATE option updates files which are already in the archive file with a newer version and will also add newly selected files that are not present in the archive file.
***FRESHEN**	The *FRESHEN option updates ONLY the files which already exist in an archive file. If the date/time of the file is newer than the date/time of the file in the archive, the file will be compressed and replace the one in the archive.
***MOVEA**	The *MOVEA (Move and Add option) option performs the *ADD option, and upon completion of a successful PKZIP command, the actual file will be deleted.
***MOVEF**	The *MOVEF (Move and Freshen option) option performs the *FRESHEN option, and upon completion of a successful PKZIP command, the actual file will be deleted.
***MOVEU**	The *MOVEU (Move and Update option) option performs the *UPDATE option, and upon completion of a successful PKZIP command, the actual file will be deleted.
***DELETE**	The *DELETE option removes entries from the archive file based upon the selection of FILES and EXCLUDE parameters. The format of the FILES and EXCLUDE parameters should be in the format of the files as seen in the archive.

TYPFL2ZP

TYPFL2ZP(*DB|*IFS)

Specifies the type of file system that contains files to be zipped. Reflected for files in parameters FILES and EXCLUDE.

***DB**	Files to be zipped are in the QSYS library file system.
***IFS**	Files to be zipped are in the IFS (Integrated File System) - Case sensitive selection.

***IFS2**	Files to be zipped are in the IFS (Integrated File System) – Non-case-sensitive selection.
***DBA**	Files to be compressed are database files in the QSYS library file system with database mode "DBSERVICE(*YES)", and the records are to be processed in arrival sequence. This is only pertinent for database files containing keys and when it is important to retain the arrival sequence of the data.
***SPL**	Files to be zipped are spool files.

TYPLISTFL

TYPLISTFL(*DB|*IFS)

Specifies the "type of files system" that will be used for the input list file and/or the output list file of selected items.

To use input list files, see parameters INCLFILE (file section list) or EXCLFILE (file exclude list). To create an output list file of the selected file items, see parameter CRTLIST.

***DB**	Files are in the QSYS library file system.
***IFS**	Files are in the IFS (Integrated File System).

VERBOSE

VERBOSE(*NORMAL|*NONE| *ALL|*MAX)

Specifies how the detail will be displayed during a PKZIP run.

The allowable values are:

***NORMAL**	Displays most informative message to show PKZIP is processing.
***NONE**	Displays only major exception information.
***ALL**	Displays all messages.
***MAX**	Used only for debugging purposes.

VPASSWORD

VPASSWORD(Archive Verify Passphrase)

Specifies a verification passphrase against the entered passphrase since the PASSWORD is not visible. This parameter is required for all encryption methods except ZIPSTD. VPASSWORD follows all the rules of PASSWORD and must match exactly to the archive passphrase entered in PASSWORD parameter or the run will be terminated. If the PASSWORD() parameter contains an inlist file, the VPASSWORD() parameter is ignored.

8 PKUNZIP Command

PKUNZIP Command Summary with Parameter Keyword Format

To decompress data from the IBM i OS command prompt screen, the command format is simply: ***PKUNZIP***.

The command prompt screen is displayed when ENTER or PF4 is pressed. The parameter keywords are displayed on this screen together with the available keyword options. If the command and parameter keywords are entered together on the command line, the required format is:

PKUNZIP keyword1(option) keyword2(option) . . . keyword*n*(option)

Keywords are delimited by spaces. The keyword “ARCHIVE” is the only positional keyword where the keyword itself is not required. Whenever the word “path” is used, its meaning depends on the file system that is being used. If IFS is used, path refers to the openness true path type. If the library systems or *DB is used, path means library/file, and then the file name refers to the member name.

```
ARCHIVE(   Archive Zip File name with path      )

AUTHCHK(   Authenticators        )           (Smartcrypt Only)
           Authenticate Type         {*FILE}
                                     {*ARCHIVE}
                                     {*ALL}
           Lookup Type               {*DB }
                                     {*LDAP}
                                     {*FILE}
                                     {*MBRSET}
                                     {*INLIST}
                                     {*SPONSOR} (Read mode Only)
           Recipient                 {Recipient String}
           Passphrase (if Private)    {Certificate passphrase}
           Required                  {*RQD }
                                     {*OPT}

AUTHPOL   (  Authenticate Filters: )           (Smartcrypt Only)
Validate Level        {*SYSTEM }
                      {*WARN }
                      {*VALIDATE}
                      {*REQUIRED}
Validate Type         {*NONE }
                      {*ALL }
                      {*ARCHIVE}
                      {*FILE}
Filters               {*SYSTEM }
                      {*ALL}
                      {*NONE}
                      {*TAMPER}
                      {*TRUSTED}
                      {*EXPIRED}
                      {*REVOKED}
                      {*NOTAMPER}
                      {*NOTTRUSTED}
                      {*NOTEXPIRED}
                      {*NOTREVOKED}

CRTLIST(   {*NONE}               )
               path/filename

CVTDATA(   External Pgm Conversion Extended Data)


CVTFLAG(   {*NONE}               )
               External Pgm Conversion Flags

CVTTYPE(   {*NONE}               )
               {*DROP}
               {*SUFFIX}

DFTDBRECLN({132}                 )
               {decimal number}

DROPPATH(  {*NONE}               )
               {*ALL}
               {*LIB}

ENTPREC(   Decryption Recipients  )          (Smartcrypt Only)
           Lookup Type               {*DB }
                                     {*FILE}
                                     {*MBRSET}
                                     {*INLIST}
           Recipient                 {Recipient String}
           Passphrase (if Private)    {Certificate passphrase}
           Required                  {*RQD }
                                     {*OPT}
EXCLFILE(  {*NONE}               )
               path/filename
```

```
EXCLUDE(   file_specification1,         )
               file_specification2,
               file_specificationn

EXDIR(     {*CURRENT}                )
               path

FACILITY ( Algorithm Facilities )                 (Smartcrypt V5R3M0 Only)
           Encryption:              {*DFT}
                                     {PKSW }
                                     {IBMSW }
                                     {PKSW_IBMSW}
                                     {IBMSW_PKSW}
           Hashing:                 {*DFT }
                                     {PKSW }
                                     {IBMSW }
                                     {PKSW_IBMSW}
                                     {IBMSW_PKSW}


FILES(     file_specification1,         )
               file_specification2,
               file_specificationn

FILETYPE(  {*TEXT}                   )
               {*BINARY}
               {*EBCDIC}
               {*DETECT}

FTRAN(       {*ISO88591}             )
             {*INTERNAL}
              Member Name

IFSCDEPAGE({*NO}                     )
               Code-page

INCLFILE(  {*NONE}                   )
               path/filename

MSGTYPE(   Outlist Details:          )
  Type         {*BOTH}
               {*PRINT}
               {*SEND}

  License Info {*NORMAL}
               {*SHORT}
               {*NONE}
               {*COPYRIGHT}

NSSRULES   (  NSS Process Settings:                 )   (Smartcrypt Only)
           NSS Classify Archive   {*SYSTEM }
                                  {*NO }
                                  {INACTIVE}
                                  {SECRET_SUITEB_REQPLUS}
                                  {SECRET_SUITEB_STRICT}
                                  {TOPSECRET_SUITEB_REQPLUS}
                                  {TOPSECRET_SUITEB_STRICT}


           NSS Check Archive State
                                  {*SYSTEM }
                                  {*NO}
                                  {*OPT}
                                  {*WARN }
                                  {*FAIL}
```

```
OVERWRITE(  {*NO}                  )
                {*YES}
                {*PROMPT}

PASSWORD(   Archive Passphrase           )

PKOVRTAPI (  Archive Tape Overrides:                        )
            Tape Device            {*TAPF }
                                   { Tape Device }
            Tape File Label        {*TAPF }
                                   {*NONE }
                                   { Tape Header Label}
            Tape Sequence Nbr      {*TAPF }
                                   { 1-16777216 }
                                   {*NEXT}
            End Of Tape Option     {*TAPF }
                                   {*LEAVE}
                                   {*REWIND}
                                   {*UNLOAD}


RSTIPSRA (  Restore Command for iPSRA Files  )

SFQUEUE (   {*DFT}                 )
                {Library/Outq }SPLUSRID (

SPLUSRID    {*DFT}                 )
                {User ID }

TRAN(         {*ISO88591}          )
              {*INTERNAL}
               Member Name

TYPARCHFL(      {*DB}                     )
                {*IFS}
                {*XDB}
                {*TAP} new

TYPE(       *VIEW                  )
                (*EXTRACT}
                {*NEWER}
                {*TEST}

TYPFL2ZP(       {*DB}                     )
                {*IFS}

TYPLISTFL(      {*DB}                     )
                {*IFS}

VERBOSE(    {*NORMAL}              )
                {*NONE}
                {*ALL}
                {*MAX}

VIEWOPT(    {*NORMAL}              )
                {*DETAIL}
                {*BRIEF}
                {*COMMENT }
                {*FNE}
                {*FNEALL}


VIEWSORT(   {*ASIS}                )
                {*DATE}
                {*DATER}
                {*NAME}
                {*NAMER}
                {*PERCENT}
                {*PERCENTR}
                {*SIZE}
                {*SIZER}
```

PKUNZIP Command Keyword Details

ARCHIVE

ARCHIVE(Archive Zip File name with path)

Specifies the path/file name or the library/file name of the archive to be extracted using PKUNZIP.

This is a required parameter.

The format is dependent upon whether the archive file will be accessed from the library file system, Integrated File System, or directly from tape.

See parameter TYPARCHFL for file system type information.

Library File System:	Format is library/file(member). If member is omitted, it will use the file name for the member.
Integrated File System (IFS):	Open system path followed by the archive file name. The path and file name can up to 256 characters and may contain embedded spaces.
TAPE (Archive directly from Tape):	If reading directly from tape, the archive file must be a Tape Device file (a file with *TAPF attributes). The tape device file will define the archive's attributes for tape usage.

AUTHCHK

Requires Smartcrypt

```
Authenticator Certificates:
    File/Archive   . . . . . . .   *FILE         *FILE, *ARCHIVE, *ALL
    LookUp Type  . . . . . . . .   *DB           *DB, *LDAP, *FILE, *MBRSET...
    Authenticator  . . . . . . .   ______________________________
    Passphrase (If Private)  . . . ______________________________
    Required . . . . . . . . . .   *RQD          *RQD, *OPT
               + for more values  _
```

Or

**AUTHCHK((*FILE *MBRSET
'pkwareCertAdmin04.pfx' (passphrase) *RQD))
AUTHCHK((*ALL *FILE
'/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer' () *RQD))
AUTHCHK((*ARCHIVE *FILE
'/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer' () *RQD))**

AUTHCHK((*FILE *DB
'EM=bill.somebody@pkware.com' () *OPT))
AUTHCHK((*FILE *INLIST 'ATEST/INLIST(ENGNEER1)' *N))

This parameter specifies that digital signature authentication processing should be performed for specific signers. Separate authentication processing may be specified for either the archive central directory or files by using multiple commands. Optionally, specific signers may be specified to authenticate against. This parameter is used in conjunction with the AUTHPOL parameters and its settings.

It is possible that more than one certificate may be returned for a single common name or email search. As a result, each one will be added to the list of validating sources.

When no specific certificates are requested, any signatories found in the archive are validated in accordance with the systems or current AUTHPOL Filters policy settings.

There are five options for AUTHCHK.

Authenticator Type File/Archive (*<u>FILE</u> |*ARCHIVE |*ALL)

This designates the type of authentication that is to be performed. Either ARCHIVE, FILE or ALL may be specified on each item, but by using ALL or archive with the *RQD option will result in error since the archive can only have one signatory. If the lookup type is *INLIST, then this option will be ignored and will pick up from the records in the inlist file.

- <u>*FILE</u> – The signed files will be authenticated with this authenticator.
- *ARCHIVE - The archive directory will be authenticated with this authenticator.
- *ALL – Both the signed files and the archive directory will be authenticated with this authenticator.

Lookup Type (*<u>DB</u> |*FILE |*LDAP |*MBRSET |*INLIST |*SPONSOR)

The lookup type would be the type of authenticator search to be used for the authenticator string to look up the public key.

- <u>*DB</u> - The authenticator string is defined to search using the certificate locator database to access the digital certificate.
- *FILE - The authenticator string is defined to read a specific file in a specific path in the IFS in order to access the digital certificate.
- *LDAP - The recipient string is defined to search using the LDAP server to access the digital certificate.
- *MBRSET - The authenticator string is defined to read this specific file from the enterprise public certificate store to access the digital certificate.
- *INLIST- The authenticator string defines a specific file that will contain one to many AUTHCHK. The TYPLISTFL parameter must specify the file type for the inlist.
- *SPONSOR - The authenticator string is the authenticating file for a sponsoring partner. This applies only for SecureZIP Partner Read mode and for *ARCHIVE.

Authenticator (The authenticator string name)

The authenticator string format depends on what was specified for the lookup type.

- If lookup type is *DB, the authenticator string will either be an email address or the common name of the certificate. This depends on the configuration setting in PKCFGSEC parameter CERTDB. To override the default selection mode, you can prefix the string with EM= for email, or CN= for the common name.

For example:

```
AUTHCHK((*FILE *DB    ‘bill.somebody@pkware.com'  () *RQD)
     (*ARCHIVE *DB    ‘CN=bill somebody'  () *RQD)
     (FILE *DB    ‘EM=bill.somebody@pkware.com'  (passphrase) *OPT))
```

- If lookup type is *FILE, the authenticator string is defined to read a specific file in a specific path of the IFS. This file should be a public key X.509 file or public key X.509 certificate with a private key file.

For example:

```
AUTHCHK((*ARCHIVE *FILE
   '/yourpath/PKWARE/Cstores/public/pkwareCertAdmin04.cer'  () *RQD))
```

The digital certificate file ‘pkwareCertAdmin04.cer’ will be in the full path '/yourpath/PKWARE/Cstores/public’.

- If type is *LDAP, the authenticator string will either be an email address or the common name of the certificate depending on the search mode configuration setting in PKCFGSEC parameter LDAP. To override the default selection mode, you can prefix the string with EM= for email address, or CN= for the common name.

For example:

```
AUTHCHK ((*ARCHIVE *LDAP    ‘bill.somebody@pkware.com'  () *RQD)
       (*FILE  *LDAP   ‘CN=bill somebody'  () *OPT)
       (*FILE  *LDAP   ‘EM=bill.somebody@pkware.com'  () *RQD))
```

- If lookup type is *MBRSET, the authenticator string is defined to read a specific file from the public certificate store and/or the private certificate store of the IFS. This file should be a public key X.509 file or public key X.509 certificate with a private key file.

For example:

```
AUTHCHK((*ALL *MBRSET 'pkwareCertAdmin04.cer'  () *RQD))
```

The digital certificate file ‘pkwareCertAdmin04.cer’ will be in the full path of the public certificate store defined in the enterprise security configuration public store (parameter CSPUB). If a passphrase is included, the file is searched for in the enterprise security configuration private store (parameter CSPRIV).

- If lookup type is *INLIST, the authenticator string defines a full file name of an input list file that contains records of AUTHCHK shortcut parameters. The type of file will exist in the QSYS library file system if TYPLISTFL(*DB) is set and will be a path file name in the IFS if TYPLISTFL(*IFS) is set. The format of the AUTHCHK shortcut parameters are defined below in the *INLIST usage section.

- If lookup type is *SPONSOR, the authenticator string is the Sponsor Auth file stored in the '…/Sponsor/Auth' folder. If the authenticator string is all numeric the name will automatically be formatted as A0000000.p7, assuming that the number is the sponsor ID number.

Passphrase

This designates the passphrase that is *required* for a private key certificate with a private key (PKCS#12 file). When a value is specified, the target must be an X.509 PKCS#12 public key certificate with the private key.

The PASSWORD value may contain blanks and is delimited by the closing right parenthesis ")" of the signing command.

Required **(*RQD|*OPT|<u>*SAME)</u>**

If *RQD, then this authenticator *must* be found during the selection, and the certificate *must* be a valid certificate with a private key, or the ZIP/UNZIP run will fail.

Usage Notes:

Passphrases are masked out in all output displays.

A local certificate store configuration is required to complete the TRUST processing of this command.

Processing is terminated if none of the requested certificates can be accessed, regardless of the "R" required flag. If multiple requests are made and at least one signature is found, processing continues normally.

For inlist that contains a passphrase to open a private certificate, make sure that the security is sufficient to only allow the owner of the certificate to have read access. Otherwise this would leave a security hole where other users could browse the passphrase.

***INLIST Usage:**

If *INLIST is defined on the AUTHCHK parameter, then the authenticator filed will be a file that Smartcrypt will read to include the authenticator. The format is very similar to the AUTHCHK parameter described above except that each line authenticator starts with "{AUTHCHK=" and is terminated by the "}" character, with the semi-colon ";" as a separator for each entry.

{AUTHCHK=Authenticator Type, Lookup Type; Authenticator; Passphrase; Required}

Authenticator *Type*	See Authenticator Type in AUTHCHK
Lookup Type	See Lookup Type in AUTHCHK excluding the INLIST
Authenticator	See Authenticator in AUTHCHK.
Passphrase	See Passphrase in AUTHCHK.
Required	See Required in AUTHCHK, but use RDQ for *RQD and OPT for *OPT.

Examples:

Sample 1: tstauth_db1.inlist.

```
{AUTHCHK=FILE;DB;EM=PKTESTDB4@nowhere.com;;RQD}
```

Sample 2: tstauth_mb2.inlist.

```
{AUTHCHK=ARCHIVE;MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

Sample 3: tstauth_mb3.inlist.

```
{AUTHCHK=ALL;MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

AUTHPOL

Requires Smartcrypt

```
Authenticate Filters:
    Validate Level     . . . . .   *SYSTEM       *VALIDATE, *WARN, *NONE...
    Validate Type      . . . . .   *ARCHIVE      *ARCHIVE, *NONE
       Filters . . . . . . . . .   *SYSTEM       *SYSTEM, *ALL, *NONE...
               + for more values
```

Or

AUTHPOL(*WARN *ARCHIVE (*SYSTEM))
AUTHPOL(*WARN *FILE (*NOTTRUSTED))
AUTHPOL(*SYSTEM *ALL (*ALL *NOTEXPIRED))

This parameter defines the processing options and filters that should apply if a signed file or signed archive is encountered.

Validate Level (*VALIDATE |*WARN |*REQUIRED |<u>*SYSTEM</u>)

The validate level specifies the type of authentication processing that should take place if a file or archive is encountered. The default is *SYSTEM and, unless it is modified, Smartcrypt will use the enterprise setting from PKCFSEC.

- ***VALIDATE** – Indicates that, when authentication takes place and a failure occurs based on the filters, the run will be considered a failure, and the message issued when the job terminates will indicate one or more errors during the run.
- ***WARN** - Indicates that when authentication is in place and a failure occurs, the failure is only considered a warning. The messages at the end of the run will not consider any failed authentications as errors.
- ***REQUIRED** – Indicates that authentication *must* take place and that, if any failure occurs based on the filters, the run will be considered a failure, and the message issued when the job terminates will indicate one or more errors occurred during the run. If the archive or file has not been signed, an error will be issued.

- ***SYSTEM –** Indicates the authentication processing that is set in the environmental setting will be used.

Validate Type (*ALL |*ARCHIVE |*FILE |*NONE)

The validate type specifies whether the file, archive, all or no authentication will take place if a file or archive has been signed. The default is *NONE, and anything other than *NONE requires the Enhanced Encryption module.

- ***ALL** - Indicates that authentication will take place for both files and/or the archive has been signed.
- ***ARCHIVE** - Indicates that only a signed archive will be authenticated.
- ***FILE** - Indicates that only the signed files will authenticated.
- ***NONE** - Indicates no authentication will take place even though a file or archive has been signed.

Filters (*SYSTEM |*ALL |*NONE |*TAMPER |*TRUSTED |*EXPIRED |*REVOKED |*NOTAMPER |*NOTTRUSTED |*NOTEXPIRED |*NOTREVOKED)

The authentication filter policies settings are defined in the enterprise security file supplied by the Smartcrypt administrator (See PKCFGSEC). These global policy settings can be revised with sub-parameter values. The variables are cumulative from the global setting.

- ***SYSTEM** – All filter policies are from the global settings.
- ***ALL** - This sub-parameter activates all levels of authentication. If followed by negating sub-levels, then all but those negating levels are activated. For example: *ALL NOTEXPIRED means that expired certificates will not cause an authentication error, but TRUST and TAMPERCHECK must both be satisfied.
- ***NONE** – Will negate all the policies.
- ***TAMPER** – This sub-parameter signifies that a verification of the data stream should be done against the digital signature.
- ***TRUSTED** – This sub-parameter signifies that the entire certificate authority chain must be validated. This includes locating the root (self-signed) certificate on the local system.
- ***EXPIRED** – This sub-parameter signifies that certificate date range validation should be performed on the certificates (including the certificate authority chain). Although the term “expired” is used, a certificate that has not yet reached its valid data range specification will fail.
- ***REVOKED** - A certificate owner may request that the issuing certificate authority declare a certificate to be revoked and thereby no longer consider that certificate to be valid. The authentication operation will fail if any of the certificates in the trust chain are found to have been revoked, or if the revocation status could not be determined
- ***NOTAMPER** – Negates the *TAMPER filter.
- ***NOTTRUSTED** – Negates the *TRUSTED filter.
- ***NOTEXPIRED** - Negates the *EXPIRED filter.

- ***NOTREVOKED** – Negates the *REVOKED filter.

CRTLIST

CRTLIST(*NONE| *path/filename*)

Specifies that PKUNZIP will create an output file with a list of entries that will be compressed based upon the selection criteria in the FILES and EXCLUDE parameters. This parameter only works with the TYPE set to *VIEW.

Use FILES and EXCLUDE to generate a listfile; use INCLFILE in a separate command to load the listfile.

See parameter TYPLISTFL for file system type information.

***NONE**	No list file will be created.
path/filename	Enter the file path and name of the file to create. The layout depends on which file system you want to create the file in.

Library File System:
The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

CVTDATA

CVTDATA(*External Program Conversion Extended Data*)

Specifies the extended data that is passed to the external program CVTNAME. When CVTFLAG is not *NONE, the contents of the parameter are passed to provide extended flexibility in controlling how the IBM i names are stored in the archive. The *System Administrator's Guide* contains more information on CVTNAME.

External Program Conversion Extended Data
Specify up to 255 bytes of unedited data which is passed to the exit program CVTNAME to assist in controlling the program logic.

CVTFLAG

CVTFLAG(*NONE|*Conversion Flags*)

Specifies the flags passed to the external program CVTNAME. These are used to control how the IBM i names are stored in the archive. The *System Administrator's Guide* contains more information on CVTNAME.

The allowable values are:

***NONE**	Conversion exit is not active.

Conversion Flags	Specify a 5-byte flag that is passed to the exit program CVTNAME to control the program logic. If the name passed back is blank, then conversion is referred back to the setting of the CVTTYPE parameter.

CVTTYPE

CVTTYPE(*NONE|*DROP|*SUFFIX)

Specifies how the files names in the archive will be converted to a file name in the IBM i library, file, and Member format. In the IBM i QSYS library system, the length of each name in the QSYS format can only be up to 10 characters. In other platforms, the file name formats (including MS/DOS) may have an extension with a period (.) separator which is not valid in the IBM i DB name. The file names in some cases may even exceed the 10-character limit. This parameter gives control over the file name conversion process.

Note: The conversion of file names may result in duplicate file names on the IBM i system. In this case, the rules for overwriting the files are in effect for duplicates (see the OVERWRITE option). If this is the case, using specific file inclusion and exclusion with multiple runs may be required to extract all of the files.

The allowable values are:

***SUFFIX**	This forces the removal of the period(.) extension and stores name truncating characters over 10 characters.
***NONE**	This leaves the IBM i name as the archive name.
***DROP**	Drops all characters after the period(.) extension separator, and stores the name truncating characters over 10.

DFTDBRECLN

DFTDBRECLN (132|Record Length)

Specifies the record length to use when creating a file in the QSYS library system. If TYPFL2ZP parameter is *DB, and the file being extracted does not exist nor does extended attribute for the record length exist, the file will be created with the record length specified in this parameter.

The allowable values are:

132	Default is record length of 132 to match previous versions.
Record Length	A decimal number from 50 to 32000.

DROPPATH

DROPPATH(*NONE|*ALL| *LIB)

Used to drop the path(s) or libraries of files that are stored in the archives, therefore only using the file names in the archive. This is used along with the keyword EXDIR where the default paths are defined when dropping the paths on files in the archive.

For example, if the file in the archive is “path1/path2/filename” (IFS) or “library/file/member” (QSYS), and if DROPPATH is *ALL, the file being extracted would be “filename” or “member”. If *LIB was used, the file being extracted would be path1/filename” or “file/member”.

The allowable values are:

***NONE**	Do not remove paths and/or libraries in the archive.
***ALL**	Remove all paths that are stored in the archive, leaving only an IFS file name or member name.
***LIB**	Remove only the first path (which in most cases could be the library).

ENTPREC

Requires Smartcrypt

```
Encryption Recipients       :
     LookUp Type  . . . . . . . .   *DB      *DB, *FILE...
     Recipient  . . . . . . . . . ______________________________________

     Passphrase               . . ______________________________________
     Required . . . . . . . . . .   *RQD          *RQD, *OPT
                + for more values   _
```

Or

ENTPREC((*MBRSET 'pkwareCertAdmin04.p12' (pw) *RQD))
ENTPREC((*FILE
  '/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.p12' (pw) *RQD))
ENTPREC((*FILE
  '/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.pfx' (‘mypassphrase’) *RQD))
ENTPREC((*DB
  ‘EM=bill.Somebody@pkware.com' (pw) *RQD))
ENTPREC((*LDAP
  ‘EM=bill.Somebody@pkware.com' (pw) *RQD))
ENTPREC((*INLIST 'ATEST/INLIST(ENGNEER1)' *N))

The decryption recipient parameter defines one to many recipients which is to be included for UNZIP process. This parameter allows 1-4 types of certificate searches to take place along with providing the ability for an include file that may contain the recipients.

The specification of this recipient ENTPREC parameter triggers decryption to take place during UNZIP processing utilizing the found recipients along with passphrases that were entered to access the private certificates.

There are four options.

Lookup Type **(*NONE |*DB |*FILE |*MBRSET |<u>*SAME)</u>**

The Lookup type is the type of recipient search to be used for the recipient string.

- ***DB** - The Recipient string is defined to search using the Certificate Locator Database to access the digital certificate.
- ***FILE** - The recipient string is defined to read a specific file in a specific path in the IFS in order to access the digital certificate.
- ***MBRSET** - The recipient string is defined to read this specific file from the enterprise public certificate store to access the digital certificate.
- ***INLIST**- The recipient string defines a specific file that will contain one to many recipients.

Recipient **(The recipient string name)**

The recipient string format depends on what was specified for the Lookup type.

- If type is *DB - The recipient string will either be an email address or the common name of the certificate. This depends on the configuration setting in PKCFGSEC parameter CERTDB. To override the default selection mode, you can prefix the string with EM= for email or CN= for the common name.

For example:

```
ENTPREC((*DB    ‘bill.Somebody@pkware.com'  (pw) *RQD)
    (*DB    ‘CN=bill Somebody'  (pw) *RQD)
    (*DB    ‘EM=bill.Somebody@pkware.com'  (pw) *RQD))
```

- If type is *FILE - The recipient string is defined to read a specific file in a specific path of the IFS. This file should be Public-key X.509 file or private-key X.509 certificate file.

For example:

```
ENTPREC((*FILE '/yourpath/PKWARE/Cstores/private/pkwareCertAdmin04.p12'
(pw) *RQD))
```

The digital certificate file ‘pkwareCertAdmin04.cer’ will be in the full path '/yourpath/PKWARE/Cstores/private.

- If type is *MBRSET - The recipient string is defined to read a specific file from private certificate store of the IFS. This file should be a private-key X.509 certificate file.

For example:

```
ENTPREC((*MBRSET 'pkwareCertAdmin04.p12'  (pw) *RQD))
```

The digital certificate file ‘pkwareCertAdmin04.p12’ will be in the full path of the private certificate store defined in the enterprise security configuration private store(parameter CSPRIV).

- If type is *INLIST- The recipient string defines a full file name of an input list file that contains records of ENTPREC shortcut parameters. The type of file will in the QSYS library file system if TYPLISTFL(*DB) is set and will be a path file name in the IFS if TYPLISTFL(*IFS) is set. The format of the ENTPREC shortcut parameters are define below in the *INLIST Usage section.

Passphrase

The passphrase is required to access private certificates.

Required (*RQD|*OPT|*SAME)

If *RQD, then this recipient MUST be found during the selection and the certificate MUST be valid or the ZIP/UNZIP run will fail.

Usage Notes:

The UNZIP process requires a X.509 private-key format certificate file to decrypt files and thus requires an inputted passphrase.

For an inlist that contains a passphrase to open a private-key certificate, make sure that the security is sufficient to allow read access only to the owner of the certificate. Otherwise other users can browse the passphrase.

***INLIST Usage:**

If *INLIST is defined on the ENTPREC parameter, then the recipient filed will be a file that Smartcrypt will read to include recipient. The format is very similar to the ENTPREC parameter describe above except each line recipient starts with "{RECIPIENT=" and is terminated by the "}" character with the semi-colon ";" as a separator for each entry.

{RECIPIENT=Lookup Type; Recipient; Passphrase; Required}

Lookup Type	See Lookup Type in ENTREC excluding the INLIST
Recipient	See Recipient in ENTREC.
Passphrase	See Passphrase in ENTREC.
Required	See Required in ENTREC, but use RDQ for *RQD and OPT for *OPT.

Sample 1: tstpriv_db4.inlist.

```
{RECIPIENT=DB;EM=PKTESTDB4@nowhere.com;PKWARE;RQD}
```

Sample 2: tstpriv_mb3.inlist.

```
{RECIPIENT=MBRSET;pktestdb3.pfx;PKWARE;RQD}
```

Sample 3: tstpubl.inlist.

```
{RECIPIENT=MBRSET;pktestdb3.p12;pw;RQD}
{RECIPIENT=MBRSET;pktestdb4.p12;pw;OPT}
```

Sample 4: tstpubl2.inlist.

```
{RECIPIENT=DB;EM=PKTESTDB3@nowhere.com;pw;RQD}
{RECIPIENT=DB;CN=PKWARE Test4;pw;OPT}
```

EXCLFILE

EXCLFILE(<u>*NONE</u>| *path/filename*)

This parameter specifies the file containing the list of files to be excluded. This can be used with or without the EXCLUDE parameter. See parameter TYPLISTFL for file system type information.

<u>*NONE</u>	No list file will be processed.
path/filename	Enter the file path and the name of the file to process. The layout depends on which file system you want the file created.

Library File System:
The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

EXCLUDE

EXCLUDE(file_specification1, file_specification2,... file_specification *n*)

Specifies the files and file specification patterns that will be excluded from the PKUNZIP run. One or more names can be specified. Each name should be in the IBM i OS file system format, such as, QSYS is library/file(member) and IFS is directory/file, and can include wildcards “*” and “?”.

Note: If TYPE(*VIEW) is being used, then the format for these names is the MS/DOS format.

The PKUNZIP program can also exclude file specifications by using the list file parameter EXCLFILE with a list of names to exclude.

Please refer to “File Selection and Name Processing” in Chapter 1 for details of file specification formatting.

The valid parameter values for the FILES keyword are as follows:

'file_specification 1'

'file_specification 2'...

'file_specification n'

EXDIR

EXDIR(*CURRENT| *path*)

If there are no paths stored in the archive file name, EXDIR specifies the default path to store the files being extracted. The path definition depends on the “file system type” in parameter TYPFL2ZP. This will happen when the files come from a PC or if the files were compressed with ***Smartcrypt***[i] using the STOREPATH(*NO) parameter.

If the “file system type” is IFS, EXDIR will be the paths defined for your IBM i open systems and the default path will be the current directory settings (issue the command DSPCURDIR to see the current directory settings).

If the “file system type” is the library file system, the path will be either a library or a library/filename. The default is *CURLIB/UNZIPPED and if the file UNZIPPED does not exist, then it is created with a record length of 132. It is best to create a default file with the record length of your choice, because if a text file is extracted with a record length greater than the file’s record length, the record will be truncated to fit the record length.

If EXDIR is coded with keyword MBR and the file system is the QSYS library system, PKUNZIP will use the member name for the file name. For example: EXDIR('newlib/MBR') and DROPPATH(*ALL) parameters are coded and the file name in archive is "mylib/myfile/mymbr", the file will be extract to the file "newlib/mymbr(mymbr)". This is only valid for TYPFL2ZP(*DB) files.

EXDIR is also used when the archive file is a GZIP archive and there is no file name stored in the archive. In this case, EXDIR becomes a required field.

***CURRENT**	Current directory for IFS or *CURLIB/UNZIPPED for the QSYS library file system.
path	Enter the path or path/path/.. in which to extract. The layout depends on the file system in which the file is to be created.

Library File System:
The format is "library/file".

Integrated File System (IFS):
The format is "path1/path2/../pathn".

FACILITY (Algorithm Facilities)

Requires Smartcrypt for IBM i V5R3M0 or above

```
Algorithm Facilities:
    Encryption:        . . . . .   *DFT          *DFT, PKSW, IBMSW...
    Hashing:           . . . . .   *DFT          *DFT, PKSW, IBMSW...
```

Or

FACILITY((IBMSW_PKSW) (*DFT))

FACILITY defines the Encryption and Hashing Algorithm API's that are available and their sequences. At this time there are only two facilities of APIs 1). PKWARE and 2). IBM Software Security.

Currently there are 2 entries for the FACILITY parameter: Encryption, and Hashing.

Encryption: (*<u>DFT</u> | PKSW | IBMSW | PKSW_IBMSW | IBMSW_PKSW)

Sets the encryption facility API to use in the run.

***DFT**	Use the encryption facility specified in the environment setting defined in the PKCFGSEC parameter FACENC
PKSW	Use PKWARE API for encryption
IBMSW	Use IBM Software API for encryption
PKSW_IBMSW	Both PKWARE API and IBM Software API are available for encryption, but use PKWARE API if available
IBMSW_PKSW	Both IBM Software API and PKWARE API are available for encryption, but use IBM Software API if available

Hashing: (*<u>DFT</u> | PKSW | IBMSW | PKSW_IBMSW | IBMSW_PKSW)

Sets the hashing facility API to use in the run.

***DFT**	Use the hashing facility specified in the environment setting defined in the PKCFGSEC parameter FACHASH.
PKSW	Use PKWARE API for Hashing.
IBMSW	Use IBM Software API for Hashing.
PKSW_IBMSW	Both PKWARE API and IBM Software API are available for Hashing, but use PKWARE API if available.
IBMSW_PKSW	Both IBM Software API and PKWARE API are available for Hashing, but use IBM Software API if available.

FILES

FILES(file_specification1, file_specification2,... file_specification *n*)

Specifies the files and file specification patterns that will be selected in the PKUNZIP process. One or more names can be specified. Each name should be in the IBM i OS file system format, such as, QSYS is library/file(member), and IFS is directory/file, and can include wildcard "*" and "?".

Note: If TYPE(*VIEW) is being used then the format for these names is the MS/DOS format.

The PKUNZIP program can also have file specification selections to include by using the list file parameter INCLFILE with a list of names to select.

Files may also be excluded. See the EXCLUDE parameter.

Please refer to “File Selection and Name Processing” in Chapter 1 for details of file specification formatting.

The valid parameter values for the FILES keyword are as follows:

'file_specification 1'

'file_specification 2'

'file_specification n'

FILETYPE

FILETYPE(*TEXT|*BINARY|*EBCDIC|*DETECT)

Specifies whether the files selected are treated as text or binary data. For text files added to an archive, trailing spaces in each line are removed, the text is converted to ASCII (based on the translation tables) by default, and a carriage return and line feed (CR/LF) are added to each line before the data is compressed into the archive. Binary files are not converted at all.

There are attributes which indicate how a file was compressed (TEXT, BINARY, or a SAVF) in the archive headers. The default setting (and recommended) is *DETECT, which analyzes the header to determine the file type. To view the attribute settings of a file, use the VIEWOPT(*DETECT).

If the file is a SAVF, then it will be processed as BINARY, regardless of any option that you select.

<u>*DETECT</u>	Uses the attribute setting that is stored in the archive to determine the file type.
***TEXT**	Specifies that the files selected are text files and translation will be performed using the translate tables specified in the TRAN option.
***BINARY**	Specifies that the files selected are binary files and no translation should be performed.
***EBCDIC**	Specifies that the files selected are text files and leaves it in EBCDIC without performing any translation. This is good only if the files are to be used on an IBM i or IBM-type mainframe. If they are unzipped to a PC file, then a translation from EBCDIC to ASCII is required.

FTRAN

FTRAN(*ISO88591 |*INTERNAL| *Member Name*)

Specifies the translation table for use in translating "file names, comments, and passphrase" from the IBM i EBCDIC character set to the character set used in the archive file (normally ASCII character set). A default internal table is predefined. See Appendix D for additional information.

***<u>ISO88591</u>**	The predefined internal table for translation. This table provides translation that is consistent with the ISO

	8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.
***INTERNAL**	To provide some compatibility to pre V8 version, *INTERNAL will use the predefined internal tables that were the default in V5 PKZIP.
membername	Specify the member name in the file PKZTABLES that will be parsed and used to translate "file names and comments" files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES. See Appendix D for information on defining translation tables.

IFSCDEPAGE

IFSCDEPAGE(*NO | *Code-Page*)

If this option is set to *NO, PKUNZIP will write IFS files with the code page that is registered for the file, or will use the default job code page if no code page is set in the file attributes. Otherwise, PKUNZIP will write IFS files with the specified code page.

Note: If files are to be extracted to a case sensitive file system, the case sensitive format of file names must be used before they can be selected.

The allowable values are:

***NO**	The PKUNZIP program will read IFS files with the code page registered for the file. This is the default.
Code-Page	The PKUNZIP program will write the IFS files with the specified code page value.

INCLFILE

INCLFILE(*NONE| *path/filename*)

This parameter specifies the file containing the list of files to be selected for including. This can be used with or without the FILES parameter. See parameter TYPLISTFL for file system type information.

***NONE**	No include list file will be processed. This is the default.
path/filename	Enter the file path and name of the file to process. The layout depends on which file system you want the file created.

Library File System:
The format is "library/file(member)".

Integrated File System (IFS):
The format is "path1/path2/../pathn/filename".

MSGTYPE

```
Outlist Details:
       Type . . . . . . . . . . .   *BOTH         *BOTH, *SEND, *PRINT
       License Info.  . . . . .   *NORMAL       *NORMAL, *SHORT, *NONE
```

Or

MSGTYPE(*SEND)
MSGTYPE(*BOTH *SHORT)
MSGTYPE(*BOTH *NONE)
MSGTYPE(*PRINT)

This parameter specifies where displayed output will be outputted and the type of licensing splash screen to provide.

Detail Type (*BOTH |*PRINT |*SEND)

Specifies where the display of messages and information should be shown. The PKZIP program can send messages that appear on the log and/or can print to stdout and stderr. If you are working interactively, stdout and stderr will show up on the dynamic screen. If working via batch, you can override stdout and stderr to print in an OUTQ or build a CL and save to an outfile.

***BOTH**	Send the information to the log with send message commands and also to stdout and stderr.
***PRINT**	Send the information to stdout and stderr.
***SEND**	Send the information to the log with send message commands.

License Info (*NORMAL |*SHORT |* NONE |*COPYRIGHT)

Specify what type of ***Smartcrypt***^{i} license and copyright information to display.

***NORMAL**	Displays all license and copyright information.
***SHORT**	Displays base licensing/copyrights.
***NONE**	Displays only registration information.
***COPYRIGHT**	Displays Copyright and trademark details from $COPYRIT file in the product distribution library.

NSSRULES

Requires Smartcrypt

```
NSS Process Settings:
    NSS Classify Archive  . . . .   *SYSTEM       *VALIDATE, *WARN, *NONE...
               + for more values
    NSS Check Archive State . . .   *SYSTEM       *NO, *WARN, *FAIL, *SYSTEM
```

Or

NSSRULES(INACTIVE)
NSSRULES (SECRET_SUITEB_REQPLUS *WARN)
NSSRULES (TOPSECRET_SUITEB_STRICT *SYSTEM)

The NSS rules parameter controls the enterprise settings for adhering to their NSS process. There are currently two option settings for NSSRULES.

NSS Classify Archive (<u>*SYSTEM</u> | *NO | INACTIVE | SECRET_SUITEB_REQPLUS | SECRET_SUITEB_STRICT | TOPSECRET_SUITEB_REQPLUS | TOPSECRET_SUITEB_STRICT)

The NSSCLASSIFY setting governs enablement of SECRET and TOP SECRET classification associated with Suite B cryptographic algorithms as specified by the National Institute of Standards and Technology (NIST) for protecting National Security Systems (NSS). Suite B includes cryptographic algorithms for encryption, digital signature, and hashing.

The default is *SYSTEM and, unless it is modified, Smartcrypt will use the enterprise setting from PKCFSEC .

The NSS Classify archive defines how the NSS Check archive state will adhere to.

- **<u>*SYSTEM</u>** - All filter policies are from the global settings.
- ***NO / INACTIVE** - No classification criteria enforcement is done.
- **SECRET_SUITEB_ REQPLUS / SS_REQP** - A restriction to algorithms and key strength specifications associated with Classification level SECRET or better are to be enforced.
- **SECRET_SUITEB_STRICT / SS_STRICT** - A restriction to algorithms and key strength specifications associated with Classification level SECRET are to be enforced exactly.
- **TOPSECRET_SUITEB_REQPLUS / TS_REQP** - A restriction to algorithms and key strength specifications associated with Classification level TOP SECRET or better are to be enforced.
- **TOPSECRET_SUITEB_STRICT / TS_STRICT** - A restriction to algorithms and key strength specifications associated with Classification level TOP SECRET are to be enforced exactly.

NSS Check Archive State (*NO | *WARN | *FAIL |<u>*SYSTEM)</u>

The NSSCHECK setting is used during VIEW, TEST or EXTRACT actions in concert with a NSSCLASSIFY specification level to be checked.

The default is *SYSTEM and, unless it is modified, Smartcrypt will use the enterprise setting from PKCFSEC.

- **<u>*SYSTEM</u>** - Indicates the authentication processing that is set in the environmental setting will be used.
- ***NO** - No check will take place.

- ***WARN** - Processing continues. Extraction is permitted to complete. A warning message AQZ0061 “Smartcrypt UNZIP ending with Warnings for <Suite B Issues>” will be returned instead of messages AQZ0037 or AQZ0038.
- ***FAIL** - Extraction processing will be terminated. The file in error will not be extracted. For all actions, the message AQZ0038 “Smartcrypt UNZIP Completed with Errors” will be returned when a mismatch occurs.

OVERWRITE

OVERWRITE(*NO|*YES|*PROMPT}

Controls how PKUNZIP reacts to files that are being extracted and the file already exists. To help prevent accidental overwriting of files, the default is *PROMPT.

The allowable values are:

***YES**	Always overwrite files. If the file exists, the file will be overwritten with no message or prompting.
***NO**	Never overwrite files. If the file already exists then the archive file will be skipped and not extracted. This is the default.
***PROMPT**	When a file being extracted already exists, PKUNZIP will issue the warning message AQZ0262 and prompt the user for the required action.

PASSWORD

PASSWORD(Archive Passphrase)

Specifies a decryption passphrase to be used for files that were encrypted to the archive with a passphrase. This passphrase may be up to 260 characters in length and is case-sensitive. All files selected for archiving will be checked for encryption using the specified passphrase. Files in the archive may have different passphrases. If so, PKUNZIP must be run once for each passphrase.

Since the passphrase in entered in EBCDIC, the translation table referenced in the FTRAN parameter is used to translate it to ASCII. Care should be taken when using the FTRAN override and when using a passphrase. To use passphrase-protected files, the same FTRAN override option is required.

If the contents of the PASSWORD() parameter starts with the key word *INLIST;, then the passphrase will be retrieved from the Inlist file defined in the PASSWORD() parameter.

*INLIST Usage

To utilize the inlist file for a passphrase, the PASSWORD parameter must start with the keyword *INLIST;. If the inlist file is not from the same file system that is set with the TYPLISTFL(*IFS/*DB) parameter, then code a *DB; or *IFS; to describe the file system where the following inlist will reside. Following the *INLIST; or *IFS;/*DB;, the file name should be specified. Using the inlist method for passphrase allows the opportunities to secure a file of a passphrase with the IBM i object authorities. For more information on INLIST see Appendix C.

Examples of PASSWORD() passphrase inlist file coding:

- **PASSWORD('*INLIST;ATEST/PPINLIST(MBR01)')**

  where TYPLISTFL(*DB) and the file is PPINLIST in library ATEST for file member MBR01

- **PASSWORD('*INLIST;*DB;ATEST/PPINLIST(MBR01)')**

  demonstrates overriding the TYPELISTFL

- **PASSWORD('*INLIST;/myroot/PKINLIST/PP01.inl')**

  where TYPLISTFL(*IFS) and the path to the inlist file is '/myroot/PKINLIST/' with stream file name of 'PP01.inl'

- **PASSWORD('*inlist;*IFS;/myroot/PKINLIST/PP01.inl'))**

  demonstrates overriding the TYPELISTFL

The passphrase inlist file must contain "PASSPHRASE={" and be terminated by the "}" character. The passphrase used will be all of the bytes that exist between the {} not including the { or the }. Null bytes and end-of-record bytes are ignored. Therefore the passphrase structure should be only on one record of an inlist file.

Example of contents of an inlist file:

```
PASSPHRASE={x12345678901234x}
```

The passphrase used will be x12345678901234x in EBCDIC. This will then be translated to ASCII using the FTRAN parameter.

Care should be taken that the file is EBCDIC and has a correct code page. It will use all bytes of data between the {}, even non-displayable bytes.

After a passphrase inlist file is set up, it should be tested with both PKZIP and the PKUNZIP command.

HEXKEY:<*display-hex value*>

This coding form (HEXKEY: in mixed case) provides for the specification of a symmetric cipher ZIP archive file protection.

HEXKEY: is used to provide a binary key for qualifying algorithms to protect files in a ZIP archive.

The key is provided in a display-hex format, where 2 hexadecimal character values comprise 1 byte (8 bits) of key information. The number of hex characters specified must match the encryption method key strength for the cipher that was used to encrypt the files.

Encryption Method	***Required Hex Characters***
3DES	42 characters (168 bits)
RC4	32 characters (128 bits)
AES128	32 characters

Encryption Method	*Required Hex Characters*
AES192	48 characters
AES256	64 characters

Example: For AES128; PASSWORD('HEXKEY:12345678123456781234567812345678')

PKOVRTAPI *new parameter*

```
New Archive Tape Overrides:
    Tape Device         . . . . .   *TAPF          Tape Device
    Tape File Label     . . . . .   *TAPF          Tape Header, *NONE
    Tape Sequence Nbr   . . . . .   *TAPF          1-16777216, *TAPF, *NEXT
    End Of Tape Option . . . . .    *TAPF          *TAPF, *REWIND, *UNLOAD...
```

Or

PKOVRTAPI(*TAPF 'ARCHIVE_TEST01' 1 *TAPF)
PKOVRTAPI(*TAPF 'ARCHIVE_TEST02' *NEXT *LEAVE)
PKOVRTAPI(*TAPF *NONE 2 *LEAVE)
PKOVRTAPI(TAP02 'ARCHIVE_TEST03' *NEXT *REWIND)

This parameter defines the override options for the tape device file specified in the archive parameter. These options are active only if TYPARCHFL(*TAP) is set to read the archive directly from a tape. When *TAP is specified, the values from the current tape device file is used. For more information on these options and overrides, refer to the CHGTAPF command.

Currently there are 4 entries for the PKOVRTAPI parameter:

(Tape Device, Tape File Label, Tape Sequence Nbr, End Of Tape Option)

Tape Device (Tape Device |*TAPF)

Overrides the DEV() parameter of the tape device file. Specifies the name of a tape device used with this device file to perform input data operations.

Tape File Label (label string |*TAPF | *NONE)

Overrides the LABEL() parameter of the tape device file. Specifies the data file identifier that is processed by this tape device file. The data file identifier is defined for standard-labeled tapes and is stored in the header label immediately before the data file that the header describes. For more details on the specification for this 17-byte option, refer to the CRTTAPF and CHGTAPF commands for tape label processing.

- ***TAPF** - Use current TAPF settings from device file.
- ***NONE** – No tape file label checking will take place. The file that will be read will depend on the tape sequence number.
- **label string** – Up to a 17 byte string for tape header label.

Tape Sequence Number (*NEXT |sequence nbr |*TAPF)

Overrides the SEQNBR() parameter of the tape device file.

This specifies the file sequence number of the data file on the tape being read. For standard-labeled tapes, this four-position file sequence number is read from the first header label of the data file.

- ***NEXT** - The file after the current setting of the tape is read.
- **Sequence number** - Specifies the file sequence number of the file being read on this tape.

End of Tape Option (*TAPF |*REWIND |*UNLOAD |*LEAVE)

Overrides the ENDOPT() parameter of the tape device file. This option specifies the operation that is automatically performed on the tape volume after the operation ends. If more than one volume is included, this parameter applies only to the last tape volume used; all other tape volumes are rewound and unloaded when the end of the tape is reached.

- ***REWIND** - The tape is rewound, but not unloaded.
- ***UNLOAD** – The tape is automatically rewound and unloaded after the operation ends.
- ***LEAVE** – The tape does not rewind or unload after the operation ends. It remains at the current position in the tape drive.

RSTIPSRA

RSTIPSRA (For iPSRA files enter a restore command)

If an iPSRA file is to be restored, RSTIPSRA should contain the appropriate restore command for the objects. To view the save command that was used to create the iPSRA file, do a TYPE(*VIEW) VIEWOPT(*ALL). This parameter should contain the restore command with no surrounding quotes. When the cursor is position to a restore command entered in the RSTIPSRA parameter, it can be prompted. If the restore command cannot pass the command pre-processor, an error will show for the restore command. Valid restore commands are: RST, RSTLIB, RSTOBJ, and RSTDLO.

SFQUEUE

SFQUEUE (*DFT |Name)

Specifies the output queue that will be used as an override when extracting spool files. If no OUTQ library is specified, it will default to *LIBL.

The allowable values are:

***DFT**	The output queue that are in the spool file attributes will be used when extracting files.
OUTQ	The specific OUTQ that will used when the spool file is extracted. It must be a valid output queue.

OUTQ Library	The library where the OUTQ resides.

SPLUSRID

SPLUSRID (<u>*DFT</u>| User ID)

The user ID to use when extracting a spool file. If *DFT is used the user ID belonging to the spool file will be used when building the spool file.

The allowable values are:

<u>*DFT</u>	Use user ID associated with spool file in the archive.
User ID	Specify a valid user ID that the new extracted spool file will belong to. It must be a valid user ID on the IBM i OS.

Note on extracting Spool Files: To create or extract a spool file with PKUNZIP, the user must have *USE authority to the API QSPCRTSP. The normal setting for the API QSPCRTSP is authority PUBLIC(*EXCLUDE). The API authority is set this way so that system administrators can control the use of this API. This API has security implications because you can create a spooled file from the data of another spooled file. To allow user to extract spool files change the API authority on a need basis.

TRAN

TRAN(<u>*ISO88591</u> |*INTERNAL| *Member Name*)

Specifies the translation table for use with translating “data” from the IBM i EBCDIC character set to the character set used in the archive file (normally the ASCII character set). A default internal table is predefined (see Appendix D).

<u>*ISO88591</u>	The predefined internal table for translation. This table provides translation that is consistent with the ISO 8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.
***INTERNAL**	To provide some compatibility to pre V8 version, *INTERNAL will use the predefined internal tables that were the default in V5 PKZIP.
Member Name	Specifies the member name in the file PKZTABLES that will be parsed and used to translate data files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES (see Appendix D for information on defining translation tables).

TYPARCHFL

TYPARCHFL(*DB |*IFS |*XDB | *TAP)

Specifies the type of file system in which the archive file will exist (see parameters ARCHIVE and TMPPATH for additional information).

***DB**	Archive files are to be in the QSYS library file system. Even though *DB is working with archive files that are in the QSYS library file system, the IFS is utilized for performance and for large file support (ZIP64). To provide an option for archive file reading utilizing exclusively the QSYS library system, use TYPARCHFL(*XDB), which will support OS400 features such as Adopt Authority.
***IFS**	Archive files are to be in the Integrated File System (IFS).
***XDB**	The archive file will be read exclusively utilizing the QSYS library file system during processing. This option will not provide large file support or ZIP64 features.
***TAP**	Archive file will be read directly from tape. The ARCHIVE parameter MUST be a tape device file that has an attribute of *TAPF. The input tape device file PKTAPEI1 is distributed with PKZIP/Smartcrypt.

TYPE

TYPE(*EXTRACT|*NEWER|*TEST |*VIEW)

The TYPE keyword specifies the type of action PKUNZIP should perform on the ZIP archive.

The possible actions are:

***VIEW**	Display output information about all files or selected files contained in an archive. This option is performed using PKUNZIP. The sequence (see *VIEWSORT) and type of list (*VIEWOPT) determines what information is displayed.
***EXTRACT**	Extracts files from the archive (please refer to the DROPPATH, CVTTYPE, TO, and EXDIR parameters for controlling the conversion of file names extracted from the archive).
***NEWER**	Extracts files in the archive that have a more recent date and time than the corresponding file on disk. If the files do not exist on disk, they will be extracted as newer. All other files will be skipped.
***TEST**	Tests the integrity of files in the archive by extracting files without writing the data. As each file is extracted, a CRC is calculated. At the end of the file the calculated

CRC is compared against the stored CRC in the archive file header to confirm that the data has not been corrupted.

TYPFL2ZP

TYPFL2ZP(*DB|*IFS)

Specifies the type of file system that contains the files to be unzipped. Reflected for files in parameters FILES and EXCLUDE.

***DB**	Files to be unzipped are in the QSYS library file system.
***IFS**	Files to be unzipped are in the IFS (Integrated File System).

TYPLISTFL

TYPLISTFL(*DB|*IFS)

Specifies the “type of files system” that will be used for the input list file and/or the output list file of selected items.

To use input list files, see parameters INCLFILE (file section list) or EXCLFILE (file exclude list). To create an output list file of the selected files items, see parameter CRTLIST.

***DB**	Files are in the QSYS library file system.
***IFS**	Files are in the IFS(integrated file system).

VERBOSE

VERBOSE(*NORMAL|*NONE| *ALL|*MAX)

Specifies how the detail will be displayed during a PKUNZIP run.

The allowable values are:

***NORMAL**	Displays most informative messages to show PKUNZIP is processing.
***NONE**	Displays only major exception information.
***ALL**	Displays all messages.
***MAX**	Used only for debugging purposes.

VIEWOPT

VIEWOPT(*NORMAL|*DETAIL|*BRIEF|*COMMENT|*FNE|*FNEALL)

Specifies the level of information produced when viewing the archive.

The allowable values are:

***NORMAL**	Shows the original file length, compression method, compressed size, compression ratio, file date and time, 32-bit CRC value, and file name for each file in the archive.
***DETAIL**	Shows very detailed technical information about each file in the archive. It will also show all extended attribute (extra data fields) information that was stored in the archive produced by PKZIP (only if the PKZIP keywords EXTRAFLD(*YES) or DBSERVICE(*YES) were specified).
***BRIEF**	Shows the original file length, file date and time, and file name for each file in the archive.
***COMMENT**	Same as the *NORMAL option, but also shows any file comments stored on a separate line after its details.
***FNE**	Shows the archive's file name encryption properties.
***FNEALL**	Shows the archive's file name encryption detail properties including the allowable recipients.

VIEWSORT

VIEWSORT(*ASIS|*DATE|*DATER|*NAME|*NAMER|*PERCENT|*PERCENTR| *SIZE|*SIZER))

Specifies the sequence of the viewing display.

The allowable values are:

***ASIS**	List the files in the sequence in which they are stored in the archive, such as, as is.
***DATE**	List the files in ascending order of the file's date & time as stored in the archive.
***DATER**	List the files in descending order of the file's date & time as stored in the archive.
***NAME**	List the files in ascending order of the file name as stored in the archive.
***NAMER**	List the files in descending order of the file name as stored in the archive.
***PERCENT**	List the files in ascending order of the compression percentage as stored in the archive.
***PERCENTR**	List the files in descending order of the compression percentage as stored in the archive.
***SIZE**	List the files in ascending order of the uncompressed file size as stored in the archive.
***SIZER**	List the files in descending order of the uncompressed file size as stored in the archive.

9 PKPGPZ “PKWARE OpenPGP ZIP” Command

Some organizations use encryption tools based on the OpenPGP standard, rather than X.509. OpenPGP uses the same basic Public Key Infrastructure principles for exchanging encrypted files, but uses a decentralized “Web of Trust” method of authenticating signatures.

Smartcrypt extracts and decrypts files that comply with the OpenPGP standard, RFC 4880. Smartcrypt can also create OpenPGP-compliant files and sign files with OpenPGP certificates. In this chapter, you’ll learn how to use Smartcrypt with OpenPGP.

PKPGPZ Requires Smartcrypt

PKPGPZ Command Summary with Parameter Keyword Format

The PKPGPZ command provides the ability to create an OpenPGP archive file format according to the RFC 4880 standard.

To create an OpenPGP file on the IBM i OS command prompt screen, the command format is simply: ***PKPGPZ***. Press Enter or PF4 to display the command prompt screen. The parameter keywords are displayed on this screen, together with the available keyword options. If the command and parameter keywords are entered together on the command line the required format is:

PKPGPZ keyword1(option) keyword2(option) . . . keyword*n*(option)

Keywords are separated by spaces. You do not need to include the “ARCHIVE” keyword; it is the only positional keyword not required. Whenever the word “path” is used in the following text, its meaning depends on the file system that is being used. If IFS is used, path refers to the open system true path type. If the library systems or *DB is used, path means library/file and then the file name refers to the member name.

Restrictions:

```
                     PKWARE OpenPGP PKPGPZ 16.0(PKPGPZ)
Type choices, press Enter.
OpenPGP Archive File:
       Archive Name  . . . . . .
Archive File Type:
        Type . . . . . . . . . .   *DB           *DB, *IFS
File to select . . . . . . . . .


Files to select Type . . . . . .   *DB           *DB, *IFS
File Data Types  . . . . . . . .   *DETECT       *DETECT *TEXT *BINARY ........
Store Path Names . . . . . . . .   *REL          *REL, *NOROOT, *NO, *YES
Compression:
               Level . . . . . .   *SUPERFAST    *SUPERFAST, *FAST, *NORMAL...
               Method  . . . . .   *DEFLATE32    *DEFLATE32, *STORE
Strong Encryption:
               Method  . . . . .   AES256        *NONE, AES128, AES192...
Archive Passphrase . . . . . . .






Verify  Passphrase . . . . . . .







Outlist Details:
        Type . . . . . . . . . .   *BOTH         *BOTH, *SEND, *PRINT
        License Info.  . . . . .   *SHORT        *NORMAL, *SHORT, *NONE...
Create a List Out file . . . . .   *NONE


File types of List Files . . . .   *DB           *DB, *IFS
Display detail levels  . . . . .   *NORMAL       *NORMAL, *NONE, *ALL, *MAX
Encryption Recipients       :
    LookUp Type  . . . . . . . .   *PGPDEF       *PGPDEF
    Handle . . . . . . . . . . .                 Handle
    Recipient  . . . . . . . . .

    Required . . . . . . . . . .   *RQD          *RQD, *OPT
               + for more values
Signing OpenPGP Keys        :
    LookUp Type  . . . . . . . .   *PGPDEF       *PGPDEF
    Handle . . . . . . . . . . .                 Handle
    Signer . . . . . . . . . . .

    Passphrase . . . . . . . . .
    Required . . . . . . . . . .   *RQD          *RQD, *OPT
Encryption Filters:
    Validate Level     . . . . .   *SYSTEM       *VALIDATE, *WARN, *SYSTEM
       Filters . . . . . . . . .   *SYSTEM       *SYSTEM, *EXPIRED...

Signing Filters:
    Validate Level     . . . . .   *SYSTEM       *VALIDATE, *WARN, *SYSTEM
       Filters . . . . . . . . .   *SYSTEM       *SYSTEM, *EXPIRED...

OpenPGP Key Ring Definitions:
    Key Ring Handle    . . . . .   *NONE         Handle
    PUB/PVT Type       . . . . .   *PUB          *PUB, *PVT
    Key Ring Engine    . . . . .   *FILE         *FILE
    Engine Name  . . . . . . . .

               + for more values
OpenPGP Rules:
    File Format  . . . . . . . .   *BINARY       *BINARY, *AARMOR, *EARMOR
    Encryption Key Select  . . .   *SYSTEM       *NONE, *LATESTVALID...
    Signing Key Select . . . . .   *SYSTEM       *NONE, *LATESTVALID...
    RFC2440 Level  . . . . . . .   *SYSTEM       *NONE, *LEVEL1
```

```
Algorithm Facilities:
    Encryption:        . . . . .   *DFT          *DFT, PKSW, IBMSW...
    Hashing:           . . . . .   *DFT          *DFT, PKSW, IBMSW...
File EBCDIC Translation mbr  . .   *ISO88591     Mbr Name
Data EBCDIC Translation mbr  . .   *ISO88591     Mbr Name
Text Line Delimiters . . . . . .   CRLF          CRLF, LFCR, CR, LF, *NONE
External Conversion Flags  . . .   *NONE         Character value, *NONE

                           Additional Parameters

IFS Code Page  . . . . . . . . .   *NO           Valid Code Page
Default DB Archive Rec. length     1024          50-32000
Insert Path  . . . . . . . . . .   *NONE

CVTNAME Extra Pass Data  . . . .

Store Directories as entry . . .   *YES          *YES, *NO
Armor File Comment . . . . . . .   *NONE
```

PKPGPZ Command Keyword Details

Advanced Encryption to use (ADVCRYPT)

When a passphrase or an OpenPGP key is specified for encryption, an encryption algorithm must be specified.

The possible encryption algorithms are:

AES256	Advanced Encryption Standard 256-bit key algorithm
AES128	Advanced Encryption Standard 128-bit key algorithm
AES192	Advanced Encryption Standard 192-bit key algorithm
3DES	Triple Data Encryption Standard.
CAST5	CAST5 encryption algorithm. (OpenPGP only)

Archive File(ARCHIVE)

This parameter specifies the name of the archive to be created.

Archive to create (archive file name with path)

Specifies the path/file name or the library/file name of the archive to be processed. If the file exists, PKPGPZ will overwrite the file, otherwise PKPGPZ will create the file for you. Depending on which file system you choose, the path or library must exist. This is a required parameter.

Note: The format of "archive file name with path" depends on whether you will be using the archive file in the library file system, or the IFS (Integrated File System) as explained in the previous section.

See parameter TYPARCHFL for file system type information.

- Library File System: Library File System Format is Library/File(Member). If Member is omitted, it will be created with the file name. If the file is not found it will be created with a default record length of 1024. (or whatever is specified in parameter DFTARCHREC). If you would create a file manually to use a larger record length, create it with no members and with parameter MAXMBRS with *NOMAX or with a high excepted limit. If the Library is not specified, the file name will be searched using the *LIBL. If the file name is not found, the file will be created in the users *CURLIB.

If a Library is specified and does not exist, the PKPGPZ program will create the Library.

- IFS (Integrated File System): Open system path followed by the archive file name.

ARCHTEXT

ARCHTEXT(*NONE| Armor File Comment)

Specifies text that will be stored in the OpenPGP Radix-64 ARMOR file as the archive's Comment record(s).

***NONE**	No new archive comment will be stored.
Armor File Comment	Up to 255 characters that are stored as the archive's armor file comments.

Compression (COMPRESS)

This parameter specifies the speed and compression level to be used when creating an OpenPGP archive.

There are two entries for the COMPRESS parameter: Level and Method.

Compression Level (*SUPERFAST |*FAST |*NORMAL |*MAX |*STORE |E0 thru E9)

The Compression Level option specifies a compression level and speed to be used. The option works in conjunction with the Compression Method option and specifies a depth of compression using a sliding scale of values. The allowable values are:

***SUPERFAST**	This is the default compression and will compress in the fastest time, but will probably compress the files by the least amount. Same as E1.
***NORMAL**	The normal compression level provides good compression amount at a reasonable speed. Same as E3.
***FAST**	This is fast compression which also provides good compression amounts. Same as E2.
***MAX**	This level provides the maximum compression, but will also take the longest in time to process. Same as E6.
***STORE**	No compression. Store will also be used if the other methods tried results in a file larger than the original. Same as E0.
E0-E9	E0 thru E9 are custom levels that can be used to try and obtain the results based on your input file and desired time and compression results.

Note: Compression levels E1 through E9 work with Deflate32.

"Maximum" is retained at level 4 to provide equivalent compression ratios with earlier releases. Higher levels may yield better compression ratios than previously obtained with "Maximum".

Compression results are data-stream dependent and produce non-linear results. When configuring a job for high volume processing, benchmarking results with a sample file may be of value to find the best balance between compression ratio and resources (elapsed and processor time). In many cases, levels 8 and 9 do not produce significant compression results over level 7.

When compression level is STORE, or E0, the compression method will be set automatically to store.

Compression Method (*DEFLATE32 |*STORE)

This option specifies the algorithm to be used when compressing a file during ZIP processing. The method works in conjunction with the Compression Level option to specify a depth of compression.

STORE performs no compression of the data. Deflate64 (using the same level control) will usually produce better compression with less processor time than Deflate32.

The allowable values are:

***DEFLATE32**	Use the Deflate 32 algorithm.
***STORE**	Store Data with no compression.

External Conversion Pass Data (CVTDATA)

Specifies the extended data passed to the external program CVTNAME. When CVTFLAG is not *NONE, the contents of the parameter is passed to provide extended flexibility in controlling how the IBM i names are stored in the archive. See Appendix C, "External Name Conversion Program" in the System Administrator's Guide for more details on CVTNAME.

External Pgm Conversion Extended Data

Specify up to 255 bytes of unedited data that is passed to the exit program CVTNAME to assist in controlling the program logic.

External Conversion Flags (CVTFLAG)

Specifies the flags passed to the external program CVTNAME. These are used to control how the IBM i names are stored in the archive. See Appendix C, "External Name Conversion Program" in the System Administrator's Guide for more details on CVTNAME.

The possible values are:

***NONE**	Conversion exit is not active.

***400**	Use the included sample CVTNAME program section that does not change the names.
Conversion Flags	Specify a 5-byte flag that is passed to the exit program CVTNAME to control the program logic. If the name passed back is blank, then conversion is referred back to the setting of the CVTTYPE parameter.

Record Delimiter (DELIM)

When compressing a text file (not binary), the DELIM parameter specifies what characters are to be appended at the end of records to serve as delimiters. The delimiter is removed from the record when it is decompressed. The allowable values are:

CRLF	This is the default selection. Specifies PKPGPZ to use the default delimiter CR-LF (x'0D0A') at the end of each text record.
CR	Appends an ASCII Carriage Return (hex 0D).
***LF**	Appends an ASCII Line Feed character (hex 0A).
LFCR	Appends an ASCII Line Feed character (hex 0A0D).

Note: MS-DOS records use CRLF for a delimiter, while UNIX records use LF.

Default Archive File Record Length(DFTARCHREC)

Specifies the record length to use when creating an archive file in the QSYS Library system. If the TYPARCHFL parameter is *DB and the archive file does not exist, the archive file will be created with the record length specified in this parameter. Care should be taken on large record lengths, as it will leave a high residual number if only one byte is use in the last record.

The allowable values are:

1024	Default is record length of 1024 to match previous versions.
Record-Length	A decimal number from 50 to 32000.

Encryption Recipients (ENTPREC)

The Encryption/Decryption Recipient parameter defines one to many recipients to include in the ZIP and UNZIP process. The specification of this Recipient ENTPREC parameter triggers encryption to take place during ZIP processing utilizing the found recipients along with any passphrase that may be entered.

There are five entries for the ENTPREC parameter: Lookup Type, Handle, Recipient, Passphrase, and Required

Lookup Type (<u>*PGPDEF</u>)

The Lookup type would be the type of recipient search that will be used for the Recipient string.

<u>*PGPDEF</u>	OpenPGP key access is defined in a Key definition (See PGPDEF).

Handle (The Keyring Handle string name)

The Keyring Handle matches up with proper supplied Key Ring (See PGPDEF).

- KeyRing Handle string name up to 8 bytes.

Recipient (The recipient string name)

The Recipient string format depends on what was specified for the Lookup type.

- If type is *PGPDEF - The Recipient string will either be an Email address, Common Name or Key ID of the OpenPGP key.

Required (*RQD|*OPT)

If *RQD, then this recipient MUST be found during the selection and the certificate MUST be valid or the ZIP/UNZIP run will fail.

- **<u>*RQD</u>** The Recipient is required to be found and valid.
- **<u>*OPT</u>** The Recipient is optional. If the Recipient cannot be found or is invalid the process will continue.

For example, if encrypting a file for John Doe, we can use either a common name, email address or the Key ID associated with his OpenPGP key:

ENTPREC((*PGPDEF *handle* 'CN=John Doe' () *RQD))
ENTPREC((*PGPDEF *handle* 'EM=john.doe@pkware.com' () *RQD))
ENTPREC((*PGPDEF *handle* 'KEYID=1234567890123456' () *RQD))

NOTE: A matching *handle* name must be specified in the PGPDEF field.

<u>Algorithm Facilities (FACILITY)</u>

FACILITY defines the Encryption and Hashing Algorithm APIs that are available and their sequences. At this time there are only two facilities of APIs

- PKWARE
- IBM Software Security.

There are two entries for the FACILITY parameter: Encryption, and Hashing.

Encryption: (*DFT |PKSW |IBMSW | PKSW_IBMSW | IBMSW_PKSW)

Defines which Encryption Facility APIs will be used during the run.

***<u>DFT</u>**	Defines that the encryption facility used is from the environment setting defined in the PKCFGSEC parameter FACENC.

PKSW	Use PKWARE API for encryption.
IBMSW	Use IBM Software API for encryption.
PKSW_IBMSW	Both PKWARE API and IBM Software API are available for encryption, but use PKWARE API if available.
IBMSW_PKSW	Both IBM Software API and PKWARE API are available for encryption, but use IBM Software API if available.

Hashing: (*DFT |PKSW |IBMSW | PKSW_IBMSW | IBMSW_PKSW)

Defines which Hashing Facility APIs will be used during the run.

***DFT**	Defines that the hashing facility used is from the environment setting defined in the PKCFGSEC parameter FACHASH.
PKSW	Use PKWARE API for hashing.
IBMSW	Use IBM Software API for hashing.
PKSW_IBMSW	Both PKWARE API and IBM Software API are available for hashing, but use PKWARE API if available.
IBMSW_PKSW	Both IBM Software API and PKWARE API are available for hashing, but use IBM Software API if available.

File to compress (FILES)

Specifies the file that will be selected in the ZIP process. One name may be specified and should be in the IBM i file system format (i.e., QSYS is library/file(member) and IFS is directory/file). For the IFS, the path and file name can be up to 256 characters and can contain embedded spaces.

File Types (FILETYPE)

Specifies whether the selected file is treated as text or binary data. When a text file is compressed, trailing spaces in each line are removed, the text is converted to ASCII (based on the translation tables) by default, and a carriage return and line feed (CR/LF) are added to each line before the data is compressed into the archive. Binary files are not converted.

The default is *DETECT, where the PKPGPZ program will try to make a determination based on the existing data type. The program will read in a portion of the data, evaluate it, and determine the appropriate process. Please note that this lowers performance time. A message will display the type used when compressing.

Use of text file options is usually faster because the PKPGPZ program has to process less data than with *BINARY. However, more processing may also take place to perform the translation.

If the file is a Save File or a Database File (with DBSERVICE(*YES)), then the file will be processed as BINARY, no matter what option is specified.

***DETECT**	The PKPGPZ program will try to determine the data type of the selected file (Text or Binary).
***TEXT**	Specifies that the selected file is a text file and translation will be performed using the translate tables specified in the TRAN option.
***BINARY**	Specifies that the selected file is a binary file and no translation should be performed.
***EBCDIC**	Specifies that the selected file is a text file and leaves it in EBCDIC without performing any translation. This is good only if the file is to be used on an IBM i or IBM-type mainframe. If it is unzipped to a PC file, then a translation from EBCDIC to ASCII would be required.
***FIXTEXT**	Specifies that the selected file is a text file with a fixed record length based on the IBM i file's record length and translation will be performed using the translate tables specified in the TRAN option. This means the compressed file will contain records with trailing spaces followed by a CR and LF. This is only valid for QSYS Library file types as files in the IFS do not contain a record length.

File EBCDIC Translation Mbr (FTRAN)

Specifies the translation table for use in translating file names, comments, and passphrases from the IBM i EBCDIC character set to the character set used in the archive file (normally ASCII character set). A default internal table is predefined. See Appendix D for additional information.

***ISO88591**	The predefined internal table for translation. This table provides translation that is consistent with the ISO 8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.
***INTERNAL**	To provide some compatibility to pre V8 version, *INTERNAL will use the internal tables that were the default in V5 PKZIP
membername	Specify the member name in the file PKZTABLES that will be parsed and used to translate "File names and comments" files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES. See Appendix D for defining translation tables.

File IFS Code Page (IFSCDEPAGE)

If this option is set to *NO, the PKPGPZ program will read IFS files using the code page that is registered for the file. Otherwise, the PKPGPZ program will read IFS files with the specified code page.

The possible values are:

***NO**	The PKPGPZ program will read IFS files with the code page registered for the file. This is the default.
Code-Value	The PKPGPZ program will read IFS files with the specified code page value.

Insert Path (ISRTPATH)

This parameter specifies a path that will be inserted in front of the file name that will be stored in the archive. This may helpful when transporting the file to another system where a self-extractor will be used. If the last position of this path is not a path separator (/), one will be added prior to inserting the path.

***NONE**	No path will be inserted. This is the default.
path1/path2/../pathn	Enter the path to insert in front of the file name that will be stored in the archive.

Outlist Details (MSGTYPE)

This parameter specifies where displayed output will be outputted and the type of licensing splash screen to provide.

There are two entries for the MSGTYPE parameter: (Detail Type, and License Info.)

Detail Type (*BOTH |*PRINT | *SEND)

Specifies where the display of messages and information should be shown. The PKPGPZ program has the ability to send messages that appear on the log and/or the ability to print to stdout and stderr. If working interactively, stdout and stderr will appear on the dynamic screen. If submitted via batch, you can override them to print in an OUTQ or build a CL and save them to an outfile.

***BOTH**	Send the information to the log with send message commands and also to stdout and stderr.
***PRINT**	Send the information to stdout and stderr.
***SEND**	Send the information to the log with send message commands.

License Info. (*NORMAL |*SHORT |*NONE |*COPYRIGHT)

Specify what type of PKZIP license and copyright information to display.

***NORMAL**	Displays all license and copyright information.
***SHORT**	Displays base licensing/copyrights.
***NONE**	Displays only registration information.
***COPYRIGHT**	Displays Copyright and trademark details from $COPYRIT file.

Archive Passphrase (PASSWORD)

Specifies an encryption passphrase for the file being compressed. This passphrase may be up to 260 characters in length and is case sensitive. The file selected for archiving will be encrypted using the specified passphrase. If a contingency key is coded for the enterprise, then the key will activate the combo-key of passphrase and certificate processing. The length range of the passphrase for Smartcrypt is defined for the enterprise settings by PKCFGSEC.

Note: There is no way to extract the passphrase used from the archive data. If the passphrase is forgotten, the file will become inaccessible.

Since the passphrase is entered in EBCDIC, the translation table referenced in the FTRAN parameter is used to translate it to ASCII. Care should be taken when using the FTRAN override and when using a passphrase. To use passphrase-protected files, the same FTRAN override option is required.

OpenPGP Keyring Definition (PGPDEF)

Specifies the 'handle' name and other settings to be used with the matching ENTPREC, and SIGNERS options.

Keyring Handle	Enter the desired 'handle' name
PUB/PVT Type	Specifies the desired key type
Keyring Engine	Specifies the access method to the OpenPGP keys.
Engine Name	Specifies the name for the appropriate access method. For example: If *FILE is specified for the Keyring Engine, the Engine Name would be the name of the file that contains the OpenPGP keys.

OpenPGP Rules (PGPRULES)

Requires Smartcrypt

PGPRULES are the base rules settings for PKPGPZ processing and consist of three options.

Example: PGPRULES(*BINARY LATESTVALID LATESTVALID *SYSTEM)

File Format (*BINARY|*AARMOR|*EARMOR)

The file format rule defines the output file's format where the default is in binary. There may be times where the file is required to be encoded into Radix-64, also known as *ASCII Armor*. This parameter will allow the resultant file to be in ASCII or EBCDIC.

***BINARY**	Produce a standard OpenPGP binary File.

***AARMOR**	Encode the standard OpenPGP binary file into an ASCII Radix-64 format. Use this parameter if the format is to be used in other OpenPGP products.
***EARMOR**	Encode the standard OpenPGP binary file into an EBCDIC Radix-64 format. Use this format only if your output file will only be used in EBCDIC environments. **Note**: We recommend that you use a record length (DFTARCHREC) of 76-80 bytes when creating EBCDIC Armor-formatted OpenPGP files.

Encryption Key Select (*NONE|*LATESTVALID|*LASTVALID|*FIRSTVALID|*FIRST |*LAST|*LATEST|*SYSTEM)

Use Encryption Key Select to restrict which OpenPGP keys are used to represent a user or organization for each encrypted file. The setting applies to the ENTPREC parameter specifying key selection from an OpenPGP keyring with either an email (EM=) or common name (CN=) selection clause. (Generic requests for an entire OpenPGP keyring or KEYID= are not affected by these settings).

When using OpenPGP public keys for recipients, it is possible to locate more than one key for a target recipient based on email or common name. By default, all matching keys that pass the ENCRYPOL policy setting will be chosen for use. However, it may be desirable to limit the encryption to a specific key based on a key's declared time range or location within the OpenPGP keyring.

Notes

- When a key has no expiration date, an implied timestamp of Mon Jan 18 22:14:07 2038 is used for comparison purposes.
- If multiple keys having the same timestamp are encountered during a time-based comparison For *FIRSTVALID, *LASTVALID or *LATESTVALID, the first valid key encountered will be used.

Options:

***<u>NONE</u>**	No matching will be performed. Every encipherment key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LATESTVALID**	The most recent encipherment having the latest expiration date (timestamp) with a valid date range including the current date/time in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LASTVALID**	The last encipherment key having a valid date range in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***FIRSTVALID**	The first encipherment key having a valid date range in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***FIRST**	The first encipherment key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.

***LAST**	The last encipherment key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LATEST**	The encipherment key having the latest expiration date (timestamp) in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***SYSTEM**	Use the enterprise setting from PKCFGSEC parameter PGPRULE for “Encryption Key Select”.

Usage Notes:

Once a qualified key is selected, it must pass the associated ENCRYPOL policy settings to be used for encryption.

When 'date range' is referred to, the specificity is actually a time stamp within a given day.

The "VALID" settings are helpful when the ENCRYPOL=EXPIRED policy is enforced and there are expired (or not yet valid) keys that might be selected. When this form of the command value is used, keys outside of the valid date range will be bypassed in the selection process.

The key found must be of a supported level or type to be used. In the event that the selection criteria and key select settings identify an unsupported or undesired key, the KEYID= search criteria should be used in the ENTPREC parameter to specify the precise key to be used.

Signing Key Select (*NONE|*LATESTVALID|*LASTVALID|*FIRSTVALID|*FIRST |*LAST|*LATEST|*SYSTEM)

Use Signing Key Select to restrict which OpenPGP key is used to represent a user or organization when generating a digital signature for an OpenPGP file. The setting applies to a SIGNERS parameter specifying key selection from an OpenPGP keyring with either an email (EM=) or common name (CN=) selection clause. (Generic requests for an entire OpenPGP keyring or KEYID= are not affected by these settings).

When using OpenPGP private-keys for signing, it is possible to locate more than one key for use as a signatory based on email or common name. By default, the first private key in the secret keyring will be chosen for use. Note that the selection process is performed without regard to private key accessibility. In other words, the password value for the SIGNERS parameter is used to access the private key after the key is selected. Smartcrypt for IBM i does not assess whether a key is accessible as part of the key selection process.

Note:

- When a key has no expiration date, an implied timestamp of Mon Jan 18 22:14:07 2038 is used for comparison purposes.
- If multiple keys having the same timestamp are encountered during a time-based comparison For *FIRSTVALID, *LASTVALID or *LATESTVALID, the first valid key encountered will be used.

Options

***NONE**	No matching will be performed. Every signing key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***FIRST**	The first signing key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LAST**	The last signing key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LATEST**	The signing key having the latest expiration date (timestamp) in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***FIRSTVALID**	The first signing key having a valid date range in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LASTVALID**	The last signing key having a valid date range in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LATESTVALID**	The most recent signing having the latest expiration date (timestamp) with a valid date range including the current date/time in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***SYSTEM**	Use the enterprise setting from PKCFGSEC parameter PGPRULE for “Signing Key Select”.

RFC2440 Level(*SYSTEM|*NONE|*LEVEL1)

When creating OpenPGP files for distribution to receiving sites that do not adhere to RFC 4880, errors may occur because the older software versions cannot interpret the files correctly. This setting allows for a range of backward compatibility to the RFC 2440 standards.

***SYSTEM**	Use the enterprise setting from PKCFGSEC parameter PGPRULE for "RFC2440 Level".
***NONE**	OpenPGP files created will adhere to RFC 4880 standards.
***LEVEL1**	If the receiving sites only support RFC 2440, *LEVEL1 will build encryption packets to RFC2440. If such sites are identified, use this setting for ZIP processing so that RFC 2440 encipherment packets (tag 9) are created instead of packet tag 18.

Usage Notes:

A valid password for the SIGNERS parameter value must be provided to access the private key for the purpose of generating a digital signature.

Once a qualified key is selected, it must pass the associated policy settings (AUTHPOL for UNZIP operations and SIGNPOL for ZIP) to be used for signing or authentication.

When 'date range' is referred to, the specificity is actually a time stamp within a given day.

The "VALID" settings are helpful when the SIGNPOL = EXPIRED policy for ZIP operations is enforced and there are expired (or not yet valid) keys that might be selected. When this form of the parameter value is used, keys outside of the valid date range will be bypassed in the selection process.

The key found must be of a supported level or type to be used. In the event that the selection criteria and key select settings identify an unsupported or undesired key, the KEYID= search criteria should be used in the SIGNERS parameter to specify the precise key to be used.

Signing Keys (SIGNERS)

This parameter identifies the private key that is to be used to digitally sign the archive directory. Signing the archive enables people who receive the archive to confirm that the archive as a whole is not changed.

There are six options for SIGNERS

Signing Type File/Archive (*ARCHIVE)

Only one signer may be specified for an archive.

***ARCHIVE**	The Archive will be signed by this private key and a signature entry will be added to the archive.

Lookup Type (*PGPDEF)

The lookup type would be the type of signer search that will be used for the signer string to lookup the private key.

***PGPDEF**	OpenPGP key access is defined in a Key definition (See PGPDEF).

Handle (The Key Ring Handle string name)

The Key Ring Handle matches up with proper supplied Key Ring (See PGPDEF).

KeyRing Handle string name up to 8 bytes.

Signer (The signer string name)

The signer string format depends on what was specified for the Lookup type.

- If type is *PGPDEF - The Recipient string will either be an Email address, Common name or Key ID of the OpenPGP key.

For example:

SIGNERS((*ARCHIVE *PGPDEF *handle* 'CN=john.doe@pkware.com' (passphrase)

***RQD))**

NOTE: A matching *handle* name must be specified in the PGPDEF field.

Passphrase (Private Cert Passphrase)

This designates the passphrase that is required for a private-key. When a value is specified, the target must be a valid OpenPGP private-key. The Passphrase value may contain blanks and is delimited by the closing right parenthesis ")" of the signing command.

Required (*RQD|*OPT)

If *RQD, then this signer MUST be found during the selection and the private key MUST be valid or the run will fail.

***RQD**	The signer is required to be found and valid.
***OPT**	The Signer is optional. If it cannot be found or is invalid the process will continue.

NOTE: It is important that the passphrase is entered in the correct case. Any variation in case or misspelling will result in a public-key certificate access attempt (which will fail for a private-key).

Passphrases will be masked out in all output displays.

Processing will be terminated if the requested private key cannot be accessed, regardless of the "R" required flag. If multiple requests are made and at least one signature is found, processing will continue normally.

<u>Signing Settings: (SIGNPOL)</u>

This parameter defines the processing options and filters that should take place if the SIGNERS parameter is used to define the archive signing private keys.

Validate Level (*VALIDATE |*WARN |*<u>SYSTEM</u>)

The validation level defines what happens if the selection of a private key for signing fails the filters.

***<u>SYSTEM</u>**	Indicates the authentication processing that is set in the environmental setting will be used.
***VALIDATE**	Indicates that when authentication takes place and a failure occurs based on the filters, the run will be considered a failure, and the message issued at the end will indicate one or more errors during the run.
***WARN**	Indicates that when authentication takes place and a failure occurs, the failure is only considered a warning. The messages at the end of the run will not consider any failed filters for signers as errors.

Filters (*EXPIRED |*NOTEXPIRED)

***EXPIRED**	The private keys used for the signing operation contains internal date ranges of validity. Note that a private key may have expired at the time that the archive is being accessed, and NOTEXPIRED may be used to continue processing.
***NOTEXPIRED**	*EXPIRED is not validated for signing keys.

Store Path (STOREPATH)

Specifies whether to store the full path and file name in the archive, or to just save the file name. If the file is an IFS file type, the path is All Directories, levels that are defined in the FILES(..) parameter. In the library system, the path is the library and the file (or member) name.

The allowable values are:

***REL**	Only the RELative path and file name will be stored (that is, No leading '/'). If an IFS absolute path was used in the file selection, the leading '/' would be removed from the file name in the archive. For example if the file is '/pkzshare/mypath1/mypath2/myfile', then it will be stored as 'pkzshare/mypath1/mypath2/myfile'.
***NOROOT**	The first node of the path will not be stored. If file type is *DB then the library will not be stored. If the file type is *IFS the first node of the path (not including a starting / if present) will not be stored. For example if the file is '/pkzshare/mypath1/mypath2/myfile', then it will be stored as 'mypath1/mypath2/myfile'.
***NO**	Store only the file name specified in the FILES(..) parameter.
***YES**	Store all path levels and the file name that is specified in the FILES(..) parameter including the leading '/' for IFS files using absolute path selection. This option is not recommended and is not valid according to the APPNOTE standards for file names in the archive.

Data EBCDIC Translation Mbr (TRAN)

Specifies the translation table for use with translating data from the IBM i EBCDIC character set to the character set used in the archive file (normally the ASCII character set). A default internal table is predefined (see Appendix D Translation Tables).

***ISO88591**	The predefined internal table for translation. This table provides translation that is consistent with the ISO 8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.

***INTERNAL**	To provide some compatibility to pre V8 version, *INTERNAL will use the internal tables that were the default in V5 PKZIP
membername	Specify the member name in the file PKZTABLES that will be parsed and used to translate "File names and comments" files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES. See Appendix D for defining translation tables.

Archive File (TYPARCHFL)

This parameter specifies the file system to create the archive and the archive constraints.

Archive File Type (*DB |*IFS)

Specifies the type of file system in which the archive file exists or that it will be created in (see parameters ARCHIVE and TMPPATH for additional information).

***DB**	Archive files are to be in the QSYS Library file system. Even though *DB is working with archive files that are in the QSYS library file system, the IFS is utilized for performance.
***IFS**	Archive files are to be in the Integrated Files System (IFS).

Files to Zip Type (TYPFL2ZP)

Specifies the type of file system for the file to be zipped.

***DB**	Files to be zipped are in the QSYS Library file system.
***IFS**	Files to be zipped are in the IFS (Integrated Files System).
***IFS2**	Files to be zipped are in the IFS (Integrated Files System) -NON Case Sensitive selection.

Display detail levels (VERBOSE)

Specifies how the detail will be displayed during a PKPGPZ run.

The possible values are:

***NORMAL**	Displays most informative message to show the PKPGPZ program processing.
***NONE**	Displays only major exception information.
***ALL**	Displays all messages.
***MAX**	Used only for debugging purposes.

Verify Passphrase (VPASSWORD)

Specifies a verification passphrase against the entered passphrase since the PASSWORD() is not visible. This parameter is required for all encryption methods except ZIPSTD. VPASSWORD() follows all the rules of PASSWORD() and must match exactly to the archive passphrase entered in PASSWORD() parameter or the run will be terminated.

10 PKPGPU “PKWARE OpenPGP UNZIP” Command

PKPGPU Requires Smartcrypt

PKPGPU Command Summary with Parameter Keyword Format

The PKPGPU command provides the ability to process an OpenPGP archive file that was created according to RFC 4880.

To extract an OpenPGP file from the IBM i OS command prompt screen, the command format is simply: PKPGPU. Press Enter or PF4 to display the command prompt screen . The parameter keywords are displayed on this screen, together with the available keyword options. If the command and parameter keywords are entered together on the command line the required format is:

PKPGPU keyword1(option) keyword2(option) . . . keyword*n*(option)

Keywords are demarcated by spaces. The keyword “ARCHIVE” is the only positional keyword where the keyword is not required. Whenever the word “path” is used, its meaning depends on the file system that is being used. If IFS is used, path refers to the open system true path type. If the library systems or *DB is used, path means library/file and then the file name refers to the member name.

```
                      PKWARE OpenPGP PKPGPU 16.0(PKPGPU)
Type choices, press Enter.
OpenPGP Archive File Name  . . .


Archive File Type  . . . . . . .   *DB           *DB, *IFS
Extract File Type  . . . . . . .   *DB           *DB, *IFS
Processing Action  . . . . . . .   *VIEW         *VIEW, *EXTRACT, *TEST
File Data Types  . . . . . . . .   *DETECT       *DETECT, *TEXT, *BINARY
Default Path . . . . . . . . . .   *CURRENT


Drop Stored Path . . . . . . . .   *NONE         *NONE, *ALL, *LIB
Archive Passphrase . . . . . . .
```

```
Overwrite existing File  . . . .   *PROMPT        *NO, *YES, *PROMPT
Outlist Details:
        Type . . . . . . . . . .   *BOTH          *BOTH, *SEND, *PRINT
        License Info.  . . . . .   *SHORT         *NORMAL, *SHORT, *NONE...
Display detail levels  . . . . .   *NORMAL        *NORMAL, *NONE, *ALL, *MAX
Default DB Create Rec. length  .   1024           50-32000
Decryption Recipients       :
    LookUp Type  . . . . . . . .   *PGPDEF        *PGPDEF
    Handle . . . . . . . . . . .                  Handle
    Recipient  . . . . . . . . .

    Passphrase . . . . . . . . .
    Required . . . . . . . . . .   *RQD           *RQD, *OPT
               + for more values

Authenticator OpenPGP Keys:
    LookUp Type  . . . . . . . .   *PGPDEF        *PGPDEF
    Handle . . . . . . . . . . .                  Handle
    Authenticator  . . . . . . .

    Required . . . . . . . . . .   *RQD           *RQD, *OPT
               + for more values

Authenticate Filters:
    Validate Level     . . . . .   *SYSTEM        *VALIDATE, *REQUIRED...
    Validate Type      . . . . .   *NONE          *NONE, *ARCHIVE
       Filters . . . . . . . . .   *SYSTEM        *SYSTEM, *EXPIRED...

OpenPGP Key Ring Definitions:
    Key Ring Handle    . . . . .   *NONE          Handle
    Pub/Pvt Type       . . . . .   *PUB           Type
    Key Ring Engine    . . . . .   *FILE          Character value, *FILE
    Engine Name  . . . . . . . .

               + for more values

OpenPGP Rules:
    Encryption Key Select  . . .   *NONE          *NONE, *LATESTVALID...
    Signing Key Select . . . . .   *NONE          *NONE, *LATESTVALID...
File EBCDIC Translation mbr  . .   *ISO88591      Mbr Name
Data EBCDIC Translation mbr  . .   *ISO88591      Mbr Name
Create a List Out file . . . . .   *NONE


External Conversion Flags  . . .   *NONE          Character value, *NONE
File types of List Files . . . .   *DB            *DB, *IFS
CVTNAME Extra Pass Data  . . . .

IFS Code Page  . . . . . . . . .   *NO            Valid Code Page
Algorithm Facilities:
    Encryption:        . . . . .   *DFT           *DFT, PKSW, IBMSW...
    Hashing:           . . . . .   *DFT           *DFT, PKSW, IBMSW...
```

PKPGPU Command Keyword Details

Archive File(ARCHIVE)

Specifies the path/file name or the library/file name of the OpenPGP archive to be processed.

Note: This is a required parameter.

The format depends on whether you will be using the archive file in the Library File System or the Integrated File System (IFS).

See parameter TYPARCHFL for file system type information.

- **Library File System**: Format is Library/File(Member). If Member is omitted, it will use the file name for the member.
- **IFS**: Open system path followed by the archive file name.

OpenPGP Signature Authentication (AUTHCHK)

This parameter specifies that OpenPGP signature authentication processing should be performed for specific signers. This parameter is used in conjunction with the AUTHPOL parameters and its settings.

It is possible that more than one OpenPGP key may be returned for a single common name or email search. As a result, each one will be added to the list of validating sources.

When no specific OpenPGP keys are requested, any signatories found in the archive are validated in accordance with the AUTHPOL() policy settings in effect.

There are four entries for AUTHCHK.

Lookup Type (*PGPDEF)

The Lookup type defines where the authenticator searches for the appropriate string to access the public key.

***PGPDEF**	The Authenticator string uses the Keyring definition to access the OpenPGP key.

Handle (The KeyRing Handle string name)

The KeyRing Handle matches up with proper supplied KeyRing (See PGPDEF).

Keyring Handle	String name up to 8 bytes.

Authenticator (The Authenticator string name)

The Authenticator string format depends on what was specified for the Lookup type.

- If lookup type is *PGPDEF, the Authenticator string will either be an email address or the common name of the OpenPGP key. This depends on the configuration setting in PKCFGSEC PGPKEYPUB and PGPKEYPVT parameters. To override the default selection mode, you can prefix the string with EM= for email, CN= for the common name or KEYID= for the key ID.

For example:

AUTHCHK((*PGPDEF *handle* 'CN=John Doe'   *RQD))
AUTHCHK((*PGPDEF *handle* 'EM=john.doe@pkware.com'   *RQD))
AUTHCHK((*PGPDEF *handle* 'KEYID=1234567890123456'   *RQD))
NOTE: A matching *handle* name must be specified in the PGPDEF field.

Required (*RQD|*OPT|*SAME)

Available values:

<u>*RQD</u>	If *RQD, then this Authenticator MUST be found during the selection and the certificate MUST be valid certificate with a private key or the ZIP/UNZIP run will fail.
***OPT**	The Authenticator is optional. If it cannot be found or is invalid the process will continue.

For example:AUTHCHK((*PGPDEF *handle* 'CN=John Doe' *RQD))
AUTHCHK((*PGPDEF *handle* 'EM=john.doe@pkware.com'   *RQD))
AUTHCHK((*PGPDEF *handle* 'KEYID=1234567890123456'   *RQD))
NOTE: A matching *handle* name must be specified in the PGPDEF field.

Usage Notes:

Processing will be terminated if none of the requested keys can be accessed, regardless of the "R" required flag. If multiple requests are made and at least one signature is found, processing will continue normally.

<u>Authenticate Filters (AUTHPOL)</u>

This parameter defines the processing options and filters that should take place if a signed archive is encountered.

There are three entries for the AUTHPOL parameter.

Validate Level (*VALIDATE |*WARN |*REQUIRED |*NONE |*SYSTEM)

The Validate Level specifies the type of authentication processing that should take place if the program encounters a signed archive . If *SYSTEM is specified, the enterprise setting from PKCFGSEC is used.

<u>*SYSTEM</u>	The enterprise setting from PKCFGSEC is used.
***VALIDATE**	Indicates that when authentication takes place and a failure occurs based on the filters, the run will be considered a failure and the message issued when the job terminates will indicate one or more errors during the run.
***WARN**	Indicates that when a failure occurs during authentication, the failure is only considered a warning. The messages at the end of the run will not consider any failed authentications as errors.
***REQUIRED**	Indicates that authentication MUST take place and if any failure occurs based on the filters, the run will be considered a failure and the message issued when the job terminates will indicate one or more errors occurred during the run. If the archive has not been signed then an error will be issued.
***NONE**	Indicates no authentication will take place even though the archive has been signed.

Validate Type (*NONE |*ARCHIVE)

The Validate Type specifies whether authentication will take place if the archive has been signed. The default is *NONE and any other value requires the Enhanced Encryption Feature.

<u>*NONE</u>	Indicates no authentication will take place even though the archive has been signed.
***ARCHIVE**	Indicates that a signed archive will be authenticated.

Filters (*SYSTEM | *TAMPER | *EXPIRED | *NOTAMPER | *NOTEXPIRED)

Available values:

<u>*SYSTEM</u>	All filter policies are from the global settings.
***TAMPER**	This sub-parameter signifies that a verification of the data stream should be done against the digital signature.
***EXPIRED**	This sub-parameter signifies that date range validation should be performed on the OpenPGP keys. Although the term "expired" is used, a key that has not yet reached its valid data range specification will fail.
***NOTAMPER**	Negates the *TAMPER filter.
***NOTEXPIRED**	Negates the *EXPIRED filter.

For example:

```
AUTHPOL(*SYSTEM *ALL *NOTEXPIRED)
AUTHPOL(*AUTH *ALL *NOTEXPIRED)
```

External Conversion Pass Data (CVTDATA)

Specifies the extended data passed to the external program CVTNAME. When CVTFLAG is not *NONE, the contents of the parameter is passed to provide extended flexibility in controlling how the IBM i names are stored in the archive. See Appendix C, “External Name Conversion Program” in the System Administrator’s Guide for more details on CVTNAME.

External Pgm Conversion Extended Data

Specify up to 255 bytes of unedited data that is passed to the exit program CVTNAME to assist in controlling the program logic.

External Conversion Flags (CVTFLAG)

Specifies the flags passed to the external program CVTNAME. These are used to control how the IBM i names are stored in the archive. See Appendix C, “External Name Conversion Program” in the System Administrator’s Guide for more details on CVTNAME.

The allowable values are:

__*NONE__	Conversion exit is not active
***400**	Use the included sample CVTNAME program section that does not change the names.
Conversion Flags	Specify a 5-byte flag that is passed to the exit program CVTNAME to control the program logic. If the name passed back is blank, then conversion is referred back to the setting of the CVTTYPE parameter.

Default DB Create Rec. length (DFTDBRECLN)

Specifies the record length to use when creating a file in the QSYS Library system. If TYPFL2ZP parameter is *DB, and the file being extracted does not exist nor does extended attribute for the record length exist, the file will be created with the record length specified in this parameter.

The allowable values are:

__132__	Default is record length of 132 to match previous versions.
Record-Length	A decimal number from 50 to 32000.

Drop Stored Path (DROPPATH)

Used to drop the path or libraries of files in the archive, therefore only using the file names in the archive. This is used along with the keyword EXDIR where the default path is defined when dropping the path on files in the archive.

For example, if the file in the archive is "path1/path2/filename" (IFS) or "Library/File/member" (QSYS), and if DROPPATH is *ALL, the file being extracted would be "filename" or "member". If *LIB was used the file being extracted would be path1/filename" or "File/member".

The allowable values are:

***__NONE__**	Do not remove paths and/or libraries in the archive.
***ALL**	Remove all paths that are stored in the archive, leaving only an IFS file name or member name.
***LIB**	Remove only the first path (which in most case could be the library).

Encryption Recipients (ENTPREC)

The Encryption/Decryption Recipient parameter defines one to many recipients to include for the ZIP and UNZIP process. The specification of this recipient ENTPREC parameter triggers encryption to take place during ZIP processing utilizing the found recipients along with any passphrase that may be entered.

There are five entries for the ENTPREC parameter:

Lookup Type (*PGPDEF)

The Lookup type defines the type of recipient search to find the appropriate recipient string.

<u>*PGPDEF</u>	OpenPGP key access is defined in a key definition (See PGPDEF).

Handle (The Keyring Handle string name)

The Keyring Handle is used to match up with proper supplied keyring (See PGPDEF).

KeyRing Handle	String name up to 8 bytes.

Recipient (The recipient string name)

The Recipient string format depends on what was specified for the Lookup type that is, CN=, EM=, KEYID=, or *ALL.

- If type is *PGPDEF - The recipient string will either be an email address, common name or Key ID of the OpenPGP key.

For example,

ENTPREC((*PGPDEF *handle* 'CN=John Doe' (password) *RQD))
ENTPREC((*PGPDEF *handle* 'EM=john.doe@pkware.com' (password) *RQD))
ENTPREC((*PGPDEF *handle* 'KEYID=1234567890123456' (password) *RQD))

NOTE: A matching *handle* name must be specified in the PGPDEF field.

Passphrase (Private Cert Passphrase)

The passphrase is required to access the private key.

Required (*RQD|*OPT)

If *RQD, then this recipient MUST be found during the selection and MUST be a valid OpenPGP key or the ZIP/UNZIP run will fail.

<u>*RQD</u>	The Recipient is required to be found and valid.
***OPT**	The Recipient is optional. If cannot be found or is invalid the process will continue.

For example,

ENTPREC((*PGPDEF *handle* 'CN=John Doe' (password) *RQD))
ENTPREC((*PGPDEF *handle* 'EM=john.doe@pkware.com' (password) *RQD))
ENTPREC((*PGPDEF *handle* 'KEYID=1234567890123456' (password) *RQD))
NOTE: A matching *handle* name must be specified in the PGPDEF field.

<u>Default Path (EXDIR)</u>

If the archive being extracted has no paths stored, EXDIR specifies the default location for the extracted files. The path definition depends on the "file system type" in parameter TYPFL2ZP. This will happen where the files may come from a PC.

- If the "file system type" is IFS, EXDIR will point to the paths defined for your IBM i open systems and the default path will be the current directory settings (issue the command DSPCURDIR to see the current directory settings).

- If the "file system type" is the Library File System, EXDIR will point to either a Library or a Library/Filename. The default is *CURLIB/UNZIPPED; if the file UNZIPPED does not exist, then it is dynamically created with a record length of 132. It is best to create a default file with the record length of your choice, because if a text file is extracted with a record length greater than the file's record length, the record will be truncated to fit the record length.
- If EXDIR is coded with keyword ?MBR and the file system is the QSYS Library system, PKPGPU will use the member name for the file name. For example: EXDIR('newlib/?MBR') and DROPPATH(*ALL) parameters are coded and the file name in the archive is "mylib/myfile/mymbr". The file will extract to "newlib/mymbr(mymbr)". This is only valid for TYPFL2ZP(*DB) files.

***<u>CURRENT</u>**	Current directory for IFS or *CURLIB/UNZIPPED for the QSYS Library file system.
path	Enter the path or path/path/.. in which to extract. The layout depends on the file system in which the file is to be created.

Library File System:

The format is "Library", "Library/File" or "Library/?MBR"

Integrated File System (IFS):

The format is "path1/path2/../pathn"

<u>Algorithm Facilities (FACILITY)</u>

FACILITY defines the Encryption and Hashing Algorithm APIs that are available and their sequences. At this time there are only two facilities of APIs:

- PKWARE
- IBM Software Security.

There are two entries for the FACILITY parameter.

Encryption: (*DFT |PKSW |IBMSW | PKSW_IBMSW | IBMSW_PKSW)

Defines which Encryption Facility APIs will be used during the run.

<u>*DFT</u>	Defines that the encryption facility used is from the environment setting defined in the PKCFGSEC parameter FACENC.
PKSW	Use PKWARE API for encryption.
IBMSW	Use IBM Software API for encryption.
PKSW_IBMSW	Both PKWARE API and IBM Software API are available for encryption, but use PKWARE API if available.

IBMSW_PKSW	Both IBM Software API and PKWARE API are available for encryption, but use IBM Software API if available.

Hashing: (*DFT |PKSW |IBMSW | PKSW_IBMSW | IBMSW_PKSW)

Defines which Hashing Facility APIs will be used during the run.

<u>*DFT</u>	Defines that the hashing facility used is from the environment setting defined in the PKCFGSEC parameter FACHASH.
PKSW	Use PKWARE API for hashing.
IBMSW	Use IBM Software API for hashing.
PKSW_IBMSW	Both PKWARE API and IBM Software API are available for hashing, but use PKWARE API if available.
IBMSW_PKSW	Both IBM Software API and PKWARE API are available for hashing, but use IBM Software API if available.

<u>File Types (FILETYPE)</u>

Specifies whether the files selected are treated as text or binary data. For text files added to an archive, trailing spaces in each line are removed, the text is converted to ASCII (based on the translation tables) by default and a carriage return and line feed (CR/LF) are added to each line before the data is compressed into the archive. Binary files are not converted at all.

The default (and recommended) setting is *DETECT, which analyzes the data to determine the file type.

***<u>DETECT</u>**	Attempt to detect the appropriate file type for UNZIP processing.
***TEXT**	Specifies that the file selected is a text file and translation will be performed using the translate tables specified in the TRAN option.
***BINARY**	Specifies that the file selected is a binary file and no translation should be performed.
***EBCDIC**	Specifies that the file selected is a text file and leaves it in EBCDIC without performing any translation. This is good only if the file is to be used on an IBM i or IBM-type mainframe. If it is unzipped to a PC file, then a translation from EBCDIC to ASCII would be required.

<u>File EBCDIC Translation Mbr (FTRAN)</u>

Specifies the translation table for use in translating File names, comments, and passphrases from the IBM i EBCDIC character set to the character set used in the archive file (normally ASCII character set). A default internal table is predefined. See Appendix D for additional information.

<u>*ISO88591</u>	The predefined internal table for translation. This table provides translation that is consistent with the ISO

	8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.
***INTERNAL**	To provide some compatibility to pre V8 versions of PKZIP, *INTERNAL will use the internal tables that were the default in V5 PKZIP
membername	Specify the member name in the file PKZTABLES that will be parsed and used to translate File names and comments files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES. See Appendix D for defining translation tables.

File IFS Code Page (IFSCDEPAGE)

If this option is set to *NO, the PKPGPU program will write IFS files with the code page that is registered for the file or the default job code page. Please note that if files are to be extracted to a case-sensitive file system, the case-sensitive format of file names must be used before they can be selected.

The possible values are:

***NO**	The PKPGPU program will read IFS files with the code page registered for the file. This is the default.
Code-Value	The PKPGPU program will write the IFS files with the specified code page value.

Outlist Details (MSGTYPE)

This parameter specifies where displayed output will be outputted and the type of licensing splash screen to provide.

There are two entries for the MSGTYPE parameter: (Detail Type, and License Info.)

Archive File Type (*BOTH |*PRINT | *SEND)

Specifies where messages and information should be displayed. The PKPGPU program has the ability to send messages that appear on the log and/or the ability to print to stdout and stderr. If working interactively, stdout and stderr will show upon the dynamic screen. If submitted via batch, you can override them to print in an OUTQ or build a CL and save them to an outfile.

***BOTH**	Send the information to the log with send message commands and also to stdout and stderr.
***PRINT**	Send the information to stdout and stderr.
***SEND**	Send the information to the log with send message commands.

License Info. (*NORMAL |*SHORT |*NONE |*COPYRIGHT)

Specify what type of license and copyright information to display.

_*_NORMAL	Displays all license and copyright information.
***SHORT**	Displays base licensing/copyrights.
***NONE**	Displays only registration information.
***COPYRIGHT**	Displays copyright and trademark details from $COPYRIT file.

Overwrite Existing Files (OVERWRITE)

Controls how the PKPGPU program reacts when extracting a file to a location where a file with the same name already exists. To help prevent accidental overwriting of files, the default is *NO. The allowable values are:

_*NO_	Never overwrite files. If the file already exists then the archive file will be skipped and not extracted. This is the default.
***YES**	Always overwrite files. If the file exists, it will be overwritten with no message or prompting.
***PROMPT**	When a file being extracted already exists, PKPGPU will issue warning message AQZ0262 and prompt the user for the required action.

Archive Passphrase (PASSWORD)

Use this option to specify a decryption passphrase if the file(s) to extract were encrypted to the archive with a passphrase. This passphrase may be up to 260 characters in length and is case-sensitive.

Since the passphrase is entered in EBCDIC, the translation table referenced in the FTRAN parameter is used to translate it to ASCII. Care should be taken when using the FTRAN override and when using a passphrase. To use passphrase-protected files, the same FTRAN override option is required.

OpenPGP Keyring Definition (PGPDEF)

Specifies the 'handle' name and other settings to be used with the matching ENTPREC, and AUTHCHK options.

Keyring Handle	Enter the desired 'handle' name
PUB/PVT Type	Specifies the desired key type
Keyring Engine	Specifies the access method to the OpenPGP keys.
Engine Name	Specifies the name for the appropriate access method.
	For example:

If *FILE is specified for the Keyring Engine, the Engine Name would be the name of the file that contains the OpenPGP keys.

OpenPGP Rules (PGPRULES)

Requires Smartcrypt

PGPRULES are the base rules settings for PKPGPU processing and consists of two options.

Example: ***PGPRULES(LATESTVALID LATESTVALID)***

Encryption Key Select (*NONE|*LATESTVALID|*LASTVALID|*FIRSTVALID|*FIRST |*LAST|*LATEST|*SYSTEM)

Use Encryption Key Select to restrict which OpenPGP keys are used to represent a user or organization for each encrypted file. The setting applies to the ENTPREC parameter specifying key selection from an OpenPGP keyring with either an email (EM=) or common name (CN=) selection clause. (Generic requests for an entire OpenPGP keyring or KEYID= are not affected by these settings).

When using OpenPGP public keys for recipients, it is possible to locate more than one key for a target recipient based on email or common name. By default, all matching keys that pass the ENCRYPOL policy setting will be chosen for use. However, it may be desirable to limit the encryption to a specific key based on a key's declared time range or location within the OpenPGP keyring.

Note:

- When a key has no expiration date, an implied timestamp of Mon Jan 18 22:14:07 2038 is used for comparison purposes.
- If multiple keys having the same timestamp are encountered during a time-based comparison For *FIRSTVALID, *LASTVALID or *LATESTVALID, the first valid key encountered will be used.

Options:

***NONE**	No matching will be performed. Every encipherment key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***FIRST**	The first encipherment key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LAST**	The last encipherment key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LATEST**	The encipherment key having the latest expiration date (timestamp) in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.

***FIRSTVALID**	The first encipherment key having a valid date range in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LASTVALID**	The last encipherment key having a valid date range in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LATESTVALID**	The most recent encipherment having the latest expiration date (timestamp) with a valid date range including the current date/time in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***SYSTEM**	Use the enterprise setting from PKCFGSEC parameter PGPRULE for “Encryption Key Select”.

Usage Notes:

Once a qualified key is selected, it must pass the associated ENCRYPOL policy settings to be used for encryption.

When 'date range' is referred to, the specificity is actually a time stamp within a given day.

The "VALID" settings are helpful when the ENCRYPOL=EXPIRED policy is desired to be enforced and there are expired (or not yet valid) keys that might be selected. When this form of the command value is used, keys outside of the valid date range will be bypassed in the selection process.

The key found must be of a supported level or type to be used. In the event that the selection criteria and key select settings identify an unsupported or undesired key, the KEYID= search criteria should be used in the ENTPREC parameter to specify the precise key to be used.

Signing Key Select (*NONE|*LATESTVALID|*LASTVALID|*FIRSTVALID|*FIRST |*LAST|*LATEST|*SYSTEM)

This option sets the system settings for PKPGPU parameter PGPRULES Signing Key Select.

Use Signing Key Select to restrict which OpenPGP key is used to represent a user or organization when generating a digital signature for an OpenPGP file. The setting applies to an AUTHCHK parameter specifying key selection from an OpenPGP keyring with either an email (EM=) or common name (CN=) selection clause. (Generic requests for an entire OpenPGP keyring or KEYID= are not affected by these settings).

When using OpenPGP private-keys for signing, it is possible to locate more than one key for use as a signatory based on email or common name. Smartcrypt for IBM i does not assess whether a key is accessible as part of the key selection process.

Note:

- When a key has no expiration date, an implied timestamp of Mon Jan 18 22:14:07 2038 is used for comparison purposes.

- If multiple keys having the same timestamp are encountered during a time-based comparison For *FIRSTVALID, *LASTVALID or *LATESTVALID, the first valid key encountered will be used.

Options:

<u>*NONE</u>	No matching will be performed. Every signing key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***FIRST**	The first signing key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LAST**	The last signing key located in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LATEST**	The signing key having the latest expiration date (timestamp) in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***FIRSTVALID**	The first signing key having a valid date range in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LASTVALID**	The last signing key having a valid date range in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***LATESTVALID**	The most recent signing having the latest expiration date (timestamp) with a valid date range including the current date/time in the designated OpenPGP keyring matching the search criteria will be counted as a viable recipient.
***SYSTEM**	Use the enterprise setting from PKCFGSEC parameter PGPRULE for “Signing Key Select”.

Usage Notes:

Once a qualified key is selected, it must pass the associated AUTHPOL policy settings to be used for encryption.

When 'date range' is referred to, the specificity is actually a time stamp within a given day.

The "VALID" settings are helpful when the AUTHPOL = EXPIRED policy is enforced and there are expired (or not yet valid) keys that might be selected. When this form of the parameter value is used, keys outside of the valid date range will be bypassed in the selection process.

The key found must be of a supported level or type to be used. In the event that the selection criteria and key select settings identify an unsupported or undesired key, the KEYID= search criteria should be used in the AUTHCHK parameter to specify the precise key to be used.

AUTH2PASSALGS (*DEFAULT|*MD5|*SHA1|*SHA256|*SHA384|*SHA512)

When authenticating older OpenPGP files NOT containing 1-pass signature algorithm information packets, this setting predisposes which hashing algorithms to run for an AUTHCHK. Algorithms MD5 and SHA1 are chosen as defaults because older OpenPGP file implementations were more likely to use these algorithms.

Note:

- Additional algorithms may be added to the configuration list if needed.
- An algorithm may be removed from the list to improve performance if no 2-pass authentication is expected for that algorithm.

Options:

***DEFAULT**	Only MD5 and SHA1 will be processed.
***MD5**	MD5 will be added to the list of algorithms to run for authentication.
***SHA1**	SHA1 will be added to the list of algorithms to run for authentication.
***SHA256**	SHA256 will be added to the list of algorithms to run for authentication.
***SHA384**	SHA384 will be added to the list of algorithms to run for authentication.
***SHA512**	SHA512 will be added to the list of algorithms to run for authentication.

AUTH2PASSMODE (*LITERAL|*BINARY|*TEXT|*BOTH)

When authenticating older OpenPGP files not containing 1-pass signature algorithm information packets, this setting predisposes which hashing mode to run for an AUTHCHK. RFC 4880 section 5.2.1 describes the modes of hashing (binary or canonical text documents) declared in the Signature Type tags.

*LITERAL processing will derive the default hashing mode based on the Literal Tag Format field (b-BINARY or t-TEXT). This assumes that the creating program of the OpenPGP file synchronized the Literal Data format with that of the Signature mode.

If a different mode is required (as indicated by the Signature Tag) for authentication to be performed, BINARY, TEXT or both may be set. Canonical Text document signature processing in RFC 4880 requires that records be formatted internally with CRLF (x'0D0A') for the purpose of hash calculations, regardless of the actual record delimiters used in the data stream. This introduces processing overhead due to the dual processing of records.

Note:

- This setting is not active when a signature algorithm/mode pre-declarative packet is present in the OpenPGP file.

Options:

***LITERAL**	Attempt to detect BINARY or TEXT from Literal Tag Format Field.

***BINARY**	Add BINARY as mode to be processed.
***TEXT**	Add TEXT as mode to be processed.
***BOTH**	Process as both BINARY and TEXT.

Data EBCDIC Translation Mbr (TRAN)

Specifies the translation table for use with translating data from the IBM i EBCDIC character set to the character set used in the archive file (normally the ASCII character set). A default internal table is predefined (see Appendix D Translation Tables").

<u>*ISO88591</u>	The predefined internal table for translation. This table provides translation that is consistent with the ISO 8859-1 definitions. This table uses the EBCDIC code page 037 and the ASCII code page 819 for translation.
***INTERNAL**	To provide some compatibility to pre V8 versions of PKZIP, *INTERNAL will use the internal tables that were the default in V5 PKZIP
membername	Specifies the member name in the file PKZTABLES that will be parsed and used to translate data files to the archive character set. The member should have the exact format of member ISO9959_1 in file PKZTABLES (see Appendix D for defining translation tables).

Archive File Type (TYPARCHFL)

Specifies the type of file system in which the archive file will exist (see parameters ARCHIVE and TMPPATH for additional information).

Archive File Type (*DB |*IFS)

***<u>DB</u>**	Archive files are to be in the QSYS library file system. Even though *DB is working with archive files that are in the QSYS library file system, the IFS is utilized for performance and for large file support (ZIP64). To provide an option for archive file reading utilizing exclusively the QSYS library system, use TYPARCHFL(*XDB), which will support IBM i features such as "Adopt Authority".
***IFS**	Archive files are to be in the Integrated File System (IFS).

Type of processing action (TYPE)

The TYPE keyword specifies the type of action the PKPGPU program should perform on the ZIP archive. The possible types are:

***EXTRACT**	Extract files from the archive (please refer to the DROPPATH, CNVTYPE, TO, and EXDIR parameters for controlling the conversion of file names extracted from the archive).
***TEST**	Tests the integrity of files in the archive by extracting files without writing the data.
***VIEW**	Will output information about all files or selected files contained in an archive.

Files to Zip Type (TYPFL2ZP)

Specifies the type of file system that contains the files to be unzipped.

***DB**	Files to be unzipped are in the QSYS Library file system.
***IFS**	Files to be unzipped are in the IFS (Integrated File System).

Display detail levels (VERBOSE)

Specifies how much detail will be displayed during a PKPGPU run.

The possible values are:

***NORMAL**	Displays most informative message to show the PKPGPU program processing
***NONE**	Displays only major exception information.
***ALL**	Displays all messages.
***MAX**	Used only for debugging purposes.

11 PKQRYCDB “Query Cert Database” Command

PKQRYCDB Requires Smartcrypt

PKQRYCDB Command Summary with Parameter Keyword Format

PKQRYCDB is a utility command to query the certificate locator database files or a certificate file in the IFS.

NOTE: OpenPGP keyrings are only supported on IFS.

Keywords are demarcated by spaces. In many cases there are multiple entries for a parameter where each entry is again demarcated by spaces. For more information about the command process reference the IBM home page for your version of the operating system.

```
                     Smartcrypt Query Cert Db  (PKQRYCDB)
Type choices, press Enter.
Processing Type  . . . . . . . .   *SUMMARY      *SUMMARY, *LEVEL1, *ALL
 File Type  . . . . . . . . . . .   *DB           *FILE, *DB, *P7B
 Certificate Type . . . . . . . .   *ALL          *PUBLIC, *PRIVATE, *ALL
 Selection Name . . . . . . . . . __________________________________________
_____________________________
 Cert Passphrase. . . . . . . . .
 Logging Level  . . . . . . . . .   *LOG          *NOLOG, *LOG, *MAXLOG
```

PKQRYCDB Command Keyword Details

RUNTYPE - Processing Type

RUNTYPE (*SUMMARY|*LEVEL1 |*SELECT|*ALL)

The processing type determines the amount of details that PKQRYCDB will display. The possible type codes are:

*SUMMARY - Shows only one line per selected item and is based on the selection type (CN= or EM=)

*LEVEL1 - Displays the common name, email address and the certificate path and file name

*SELECT - Displays a display file of certificates based on the selection type. The items can be browsed or selected for a detail display of the certificates. If the certificate dates have expired, the dates will be highlighted.

*ALL - Displays a complete set of details for each certificate; could be 20-40 lines per file

FTYPE - File Type

FTYPE (*FILE| *DB | *P7B | *CRL | *PKCS12)

The file type determines the type of path/file name in the parameter FNAME.

If *DB is selected, PKQRYCDB will search the database based on the contents of the FNAME. For example, CN=Bill* will search for all certificates with a common name that starts with Bill regardless of upper case or lower case.

If *FILE is selected, then FNAME should be a very specific certificate file (full path included).

*P7B will read a specific file that should be in a P7B format. It will then do a detailed display for the contents of the P7B certificate store.

*CRL will read a specific file that should be in a CRL format. It will then do a detailed display for the contents of the CRL file.

*PKCS12 will read a specific file that should be in a PKCS12 format. Password is required. It will then do a detailed display for the contents of the PKCS12 file.

CTYPE - Certificate Type

CTYPE (*ALL| *PUBLIC | *PRIVATE)

CTYPE specifies the type of certificates, private or public, that will be processed in this run.

*ALL will process both public and private certificates.

*PUBLIC specifies that only public certificates should be processed. No passphrase should be supplied.

*PRIVATE indicates that only private-key certificates should be processed and requires that a passphrase be entered.

FNAME - File Name

FNAME (Path/File name)

If FTYPE is *DB, the FNAME contents will be the selection criteria for the certificate locator database. It should contain the prefix of the field to select, such as CN= for common name and EM= for email address. Selection is not case-sensitive. If the selection ends in an asterisk (*), a generic selection is made for all certificates starting with the selection criteria.

If FTYPE is *FILE, the contents of FNAME contains the IFS file that will used to query the certificate contents. Specify the full path and file name of the specific certificate file.

PASSWORD - Certificate Passphrase

PASSWORD (Certificate Private Key Passphrase)

Processing the private key certificate with RUNTYPE(*ALL) requires the passphrase used when the certificate was exported to open and gather the contents. The passphrase is used only to open the certificate to gather the database data; it is not stored or saved. The certificate is not altered in any way.

LOGLVL - Logging Level

LOGLVL (*LOG|*NOLOG |*MAXLOG)

Specifies the level of logging (printing/viewing) used during a PKQRYCDB run. LOGLVL(*NOLOG) shows only a minimal amount of information. LOGLVL(*MAXLOG) shows more details, with some detail useful only for problem determination.

Sample Displays

Request RUNTYPE(*SUMMARY) to generate and display a report containing additional information about the certificate.

➔ **PKQRYCDB RUNTYPE(*SUMMARY) FNAME('cn=will*')**

```
PKQRYCDB QUERY Smartcrypt Cert DataBase starting------2004/11/16 07:37:28
PKQRYCDB Start Search Summary for <cn=will*>
 Public Key CN=William S. Somebody
 Public Key CN=William Somebody
 Public Key CN=William Somebody
Private Key CN=William Somebody
 Public Key CN=William Somebody


PKQRYCDB Run Totals:
   Total Records In Error  =0
   Total Records Processed =5
PKQRYCDB Scan ending------
```

Request RUNTYPE(*Level1) to generate and display a report containing additional information about the certificate.

➔ **PKQRYCDB RUNTYPE(*LEVEL1) FNAME('cn=will*')**

```
PKQRYCDB QUERY Smartcrypt Cert DataBase starting------2004/11/16 07:39:34
PKQRYCDB Start Search Level 1 for <cn=will*>
 Public Key CN=William S. Somebody
            EM=sombody@worldnet.att.net
            FN=William S. Somebody
            File </yourpath/PKWARE/Cstores/public/williamsSomebody.cer>
 Public Key CN=William Somebody
            EM=bill.Somebody@pkware.com
            FN=William Somebody
            File </yourpath/PKWARE/Cstores/public/billSomebody03.cer>
 Public Key CN=William Somebody
            EM=bill.Somebody@pkware.com
            FN=William Somebody
            File </yourpath/PKWARE/Cstores/public/bill_Somebody2003.cer>
Private Key CN=William Somebody
            EM=bill.Somebody@pkware.com
            FN=William Somebody
            File </yourpath/PKWARE/Cstores/private/billSomebody03.pfx>
 Public Key CN=William Somebody
            EM=bSomebody@pkware.com
            FN=William Somebody
            File </yourpath/PKWARE/Cstores/public/billSomebody.cer>


  PKQRYCDB Run Totals:
     Total Records In Error  =0
     Total Records Processed =5
  PKQRYCDB Scan ending------
```

Request RUNTYPE(*ALL) to generate and display a report containing additional information about the certificate.

➔ **PKQRYCDB RUNTYPE(*ALL) FTYPE(*FILE) FNAME('/yourpath/PKWARE/Cstores/public/billSomebody03.cer')**

```
PKQRYCDB QUERY Smartcrypt Cert DataBase starting------2004/11/16 07:43:50


 ---------------------------------------------------------
 Public Key Found File </yourpath/PKWARE/Cstores/public/billSomebody03.cer>
            CN=William Somebody
            EM=bill.Somebody@pkware.com
            FN=William Somebody

--- Certificate ---
William Somebody
Subject:
   O=VeriSign, Inc.
   OU=VeriSign Trust Network
   OU=www.verisign.com/repository/RPA Incorp. by Ref.,LIAB.LTD(c)98
   OU=Persona Not Validated
   OU=Digital ID Class 1 - Microsoft Full Service
   CN=William Somebody
   E=bill.Somebody@pkware.com
Issuer:
   O=VeriSign, Inc.
   OU=VeriSign Trust Network
   OU=www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98
   CN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
SerialNumber:
   3F55 2A91 2B5A 9F9B 46E0 D8A0 96DB DDAB
NotBefore:
   Mon Jul 21 19:00:00 2003
NotAfter:
   Wed Jul 21 18:59:59 2004
SHA-1 Hash of Certificate:
   D5 CE FF A5 72 EF B6 53 EA 75 F7 CA 2E 01 85 7B
   65 7C B8 E7
Public Key Hash:
   6E 16 CF EF FA A0 99 25 2B 79 DE E6 23 C7 D7 42
   80 82 F3 E4
End Entity


PKQRYCDB Run Totals:
   Total Records In Error  =0
   Total Records Processed =1
PKQRYCDB Scan ending------
```

The following table explains the fields of the certificate details in the display.

Heading	***Description***
Subject	Information about the entity to whom the certificate was issued
Issuer	Information about the entity that issued the certificate
Serial Number	Serial number of the certificate
NotBefore/NotAfter	Date range for which the certificate is valid
SHA-1 Hash of Certificate	The SHA-1 algorithm hash, or “thumbprint,” of the certificate
Public Key Hash	The hash, or “thumbprint,” of the public key
Key Usage	Key usage flags that determine how the certificate was intended to be used
Fingerprint/Thumbprint	The fingerprint is a unique string of characters that exactly identifies a key.
OpenPGP Key ID	(OpenPGP keyring only) The KeyID is similar to the Fingerprint. However the KeyID only contains the last 8 characters of the fingerprint. Most of the time it is possible to identify a key with only the KeyID, but occasionally two keys may have the same ID.

The public key hash value is the prime key used in the local certificate store index.

The *Issuer* fields are composed of several X.509 subfields. The exact set varies. The following table describes some of the most commonly used.

Code	***Description***
O	Organization
OU	Organizational Unit
CN	Common Name
E or **EM**	Email address
C	Country
ST	State or Province
L	Locality or City

The common name (CN) and email (E) fields can be searched to identify recipients.

Request RUNTYPE(*SELECT) to generate a browse screen containing additional information about the certificate. This provides the ability to fold and unfold for more information. To display details as shown above, enter a 5.

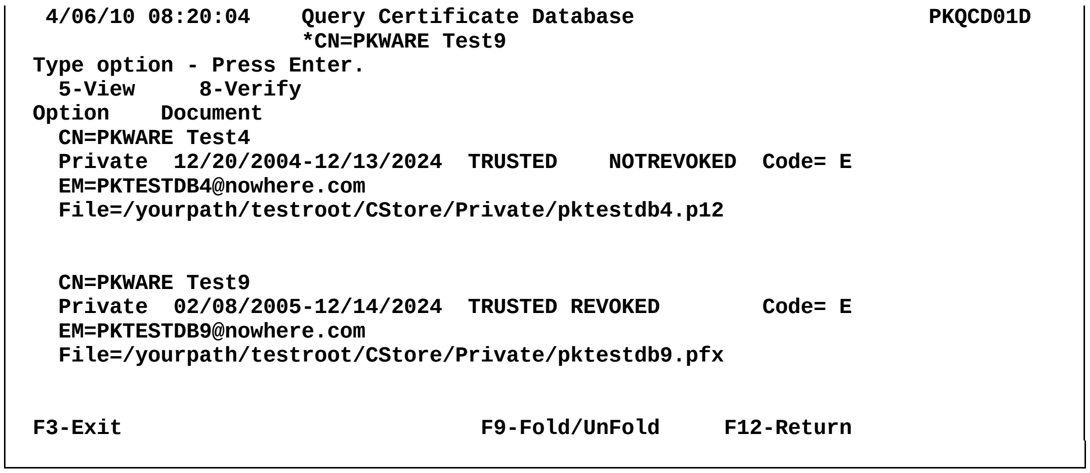
➔ **PKQRYCDB RUNTYPE(*SELECT) FNAME('cn=P*')**

Folded

```
  4/06/10 08:20:04    Query Certificate Database                       PKQCD01D
                      *CN=PKWARE Test9
 Type option - Press Enter.
   5-View     8-Verify
 Option    Document
_  CN=PKWARE Test1
_  CN=PKWARE Test3
_  CN=PKWARE Test3
_  CN=PKWARE Test4
_  CN=PKWARE Test4
_  CN=PKWARE Test9

F3-Exit                            F9-Fold/UnFold     F12-Return
```

F9 to Unfold

```
  4/06/10 08:20:04    Query Certificate Database                       PKQCD01D
                      *CN=PKWARE Test9
 Type option - Press Enter.
   5-View     8-Verify
 Option    Document
   CN=PKWARE Test1
   Public   04/14/2004-04/13/2024  NOTTRUSTED NOTREVOKED  Code= CES
   EM=PKTESTDB1@nowhere.com
   File=/yourpath/testroot/CStore/Public/pktestdb1.cer


   CN=PKWARE Test3
   Public   12/20/2004-12/13/2024  TRUSTED    NOTREVOKED  Code= E
   EM=PKTESTDB3@nowhere.com
   File=/yourpath/testroot/CStore/Public/pktestdb3.crt

                                                                        +
 F3-Exit                            F9-Fold/UnFold     F12-Return


  4/06/10 08:20:04    Query Certificate Database                       PKQCD01D
                      *CN=PKWARE Test9
 Type option - Press Enter.
   5-View     8-Verify
 Option    Document
   CN=PKWARE Test3
   Private  12/20/2004-12/13/2024  TRUSTED    NOTREVOKED  Code= E
   EM=PKTESTDB3@nowhere.com
   File=/yourpath/testroot/CStore/Private/pktestdb3.p12


   CN=PKWARE Test4
   Public   12/20/2004-12/13/2024  TRUSTED    NOTREVOKED  Code= E
   EM=PKTESTDB4@nowhere.com
   File=/yourpath/testroot/CStore/Public/pktestdb4.crt



                                                                         +

 F3-Exit                            F9-Fold/UnFold     F12-Return
```



```
 4/06/10 08:20:04    Query Certificate Database                       PKQCD01D
                     *CN=PKWARE Test9
Type option - Press Enter.
  5-View     8-Verify
Option    Document
  CN=PKWARE Test4
  Private  12/20/2004-12/13/2024  TRUSTED    NOTREVOKED  Code= E
  EM=PKTESTDB4@nowhere.com
  File=/yourpath/testroot/CStore/Private/pktestdb4.p12


  CN=PKWARE Test9
  Private  02/08/2005-12/14/2024  TRUSTED REVOKED        Code= E
  EM=PKTESTDB9@nowhere.com
  File=/yourpath/testroot/CStore/Private/pktestdb9.pfx


F3-Exit                            F9-Fold/UnFold     F12-Return
```

12 Processing with GZIP

What Is GZIP?

GNU Zip is a different standard for handling compressed file data in an archive. Support for the GZIP standard can be found in various utilities for many platforms. This format is not compatible with ***Smartcrypt****i* archives; however, ***Smartcrypt****i* provides limited support for GZIP archives (Information regarding RFC processes for information interchange with regard to GZIP can be found at www.faqs.org/rfcs).

RFC 1951 is the specification that describes the DEFLATE compressed data format that is to be used with GZIP archives. ***Smartcrypt****i* creates a compression stream that is compatible with this format.

RFC 1952 describes the GZIP archive format specifications. Differences from ***Smartcrypt****i* archives include:

- All GZIP file names must be represented in lower case.
- Both binary and text data are supported by GZIP; however, the LATIN-1 translation table is the defined standard for EBCDIC/ASCII file name translation (ISO 8859-1).

Why Use GZIP?

GZIP may be useful when doing file exchanges to a platform only having a GZIP support utility.

Although GZIP has an almost limitless capacity, it has other significant limitations that make it less attractive than ***Smartcrypt****i* for most applications.

- GZIP lacks a "directory" of the files contained within it. In addition, files contained within a GZIP archive can only be found in a serial fashion. (GZIP and ZIP have different nomenclature. Whereas a ZIP archive stores "files", these data entities are known as "members" in a GZIP archive.)
- The file information controls provided in GZIP archives cannot be fully reported on until the entire data stream is decompressed.

- The GZIP format may not be recognized by other products providing ZIP archive support, and thereby restricts its cross-platform usefulness.

PKZIP and *Smartcrypt for z/OS* Implementation Notes for GZIP

The DEFLATE compression algorithm used in GZIP is similar to the compression logic used in ***Smartcrypt**^{i}* archives. The archive format is compatible with GZIP processes running on other platforms, although extensions provided by ***Smartcrypt**^{i}* may not be supported by other utilities.

The standard GZIP archive format maintains a header entry at the beginning that describes the name of the file and a timestamp. A CRC integrity value is also maintained, however, this value is stored at the end of the file along with the original size of the input file.

GZIP Restrictions

- The ***Smartcrypt**^{i}* implementation for GZIP is restricted to one file within an archive. For this reason, only the ADD Action for a new archive is supported. Do not attempt to FRESHEN an existing file within an archive, add additional files, or delete a file from an archive.
- Only the first file in a GZIP archive from another platform will be processed by UNZIP processing. For this reason, when creating GZIP archives on other platforms with IBM i as the target system, only place one file in each GZIP archive file.
- An existing archive must be processed in accordance with its archive type, such as, ***Smartcrypt**^{i}* or GZIP. For example, an existing ***Smartcrypt**^{i}* archive cannot have GZIP data appended to it. A message will be issued and processing will be terminated if this rule is not followed.
- VIEW processing will not report the CRC or file size information because of the way GZIP archives hold the information.
- Only COMPRESS(*STORE) and COMPRESS(*TERSE) are not part of the GZIP standard, and is therefore ignored by the compression engine.
- The GZIP standard does not support strong encryption.
- The GZIP standard does not support digital signatures.
- Parameter FTRAN is not valid for GZIP because file names have to be held in the ISO 8859-1(LATIN-1) character set..

GZIP Extensions

- As a proprietary extension, standard (96 bit) password encryption support is provided beyond the RFC standard.
- File attributes can be stored in the GZIP archive (just as they are in a ***Smartcrypt**^{i}* archive) so that the file can be reconstructed during EXTRACT processing.
- File name control parameters such as STOREPATH, CVTTYPE or CVTFLAG may be used.

- During EXTRACT processing, if the GZIP archive does not contain a file name (not required by GZIP specifications), then a file name is constructed with the contents of the EXDIR parameter. If EXDIR was not specified the run will end in error.
- Although the default specification for GZIP processing is to handle data as *BINARY, ***Smartcrypt**^{i}* will use the FILETYPE parameter with *DETECT or *TEXT processing.
- Although the GZIP standard does not support directory levels in the file name, many products (including ***Smartcrypt**^{i}*) support this as an extension.
- Although the timestamp in the archive is in UNIX-format and is by specification to be UTC, ***Smartcrypt**^{i}* honors the TIMESTAMP command.

Processing GZIP Archives

In general, a GZIP archive must be processed only in GZIP mode and with only one GZIP "member." When creating a GZIP archive, specify GZIP(*YES) in the command stream. UNZIP processing in ***Smartcrypt**^{i}* automatically detects the GZIP header and processes accordingly.

Special Note on GZIP Passphrases

GZIP standard processing (RFC 1952) does not normally allow a passphrase to be placed on a GZIP archive. ***Smartcrypt**^{i}* does allow this feature, but its use may cause compatibility issues with other platforms. ***PKZIP for MVS*** does use the same passphrase standard, so GZIP archives with passphrases can be exchanged between ***Smartcrypt for IBM i**, **SecureZIP for zSeries***, and ***PKZIP for MVS***. Because GZIP archives that are created with a passphrase with ***SecureZIP**^{i}* or ***PKZIP for MVS™*** are not part of the GZIP standards, these files will probably appear to be corrupt on other platforms.

Sample GZIP Processing

Compressing a file

The following example shows how to compress a file into a GZIP archive. The PKZIP command is used to compress data into an archive. To select the GZIP format for the resulting archive, you must use the GZIP(*YES) option with the PKZIP command:

PKZIP ARCHIVE('MYLIB1/MYARCHFIL(GZ01)') FILES('TESTLIB1/FILE1TXT') TYPE(*ADD) GZIP(*YES)

The command above will compress the text file TESTLIB1/FILE1TXT into the archive MYLIB1/MYARCHFIL(GZ01) in GZIP format. The archive must not already exist or an error message will be generated and the operation will fail.

The output from the above command should look like the following:

```
File MYARCHFIL created in library MYLIB1.

Scanning files for match  ...
File MYARCHFIL in library MYLIB1 with member GZ01 not found.
```

```
Found  1 matching files
Member GZ01 added to file MYARCHFIL in MYLIB1.
Member GZ01 removed from file MYARCHFIL in MYLIB1.
Member PZ3AF2447F added to file MYARCHFIL in MYLIB1.
Compressing TESTLIB1/FILE1TXT(FILE1TXT) in TEXT mode
Add  TESTLIB1/FILE1TXT/FILE1TXT  --   Deflating (31%)
Member PZ3AF2447F renamed to member GZ01.
Member GZ01 file MYARCHFIL in MYLIB1 changed.
PKZIP Compressed 1 files in GZIP Archive MYLIB1/MYARCHFIL(GZ01)
PKZIP Completed Successfully
```

13 Processing with OpenPGP

Overview: OpenPGP vs. X.509

Some organizations use encryption tools based on the OpenPGP standard, rather than X.509. OpenPGP uses the same basic Public Key Infrastructure principles for exchanging encrypted files, but uses a decentralized “Web of Trust” method of authenticating signatures.

Smartcrypt extracts and decrypts files that comply with the OpenPGP standard, RFC 4880. Smartcrypt can also create OpenPGP-compliant files and sign files with OpenPGP certificates. In this section, you’ll learn more about the OpenPGP standard, and how to use Smartcrypt with OpenPGP.

Note: This guide assumes you have some knowledge of, and experience working with OpenPGP files, and does not include a comprehensive introduction to OpenPGP.

As described in “Public-Key Infrastructure and Digital Certificates” in chapter 2, "Introduction to Data Security,” the X.509 standard relies on a hierarchical “trust chain” model, where an individual digital signature is issued by an intermediate Certificate Authority (CA), which is assumed to have received enough documentation to determine that an individual is who he says he is. The intermediate CA’s certificate gets its certificate, in turn, from a Root CA. Each certificate says who issued it, and theoretically if you question the authenticity of a certificate, you can find the documentation presented to the original CA.

OpenPGP certificates are typically created by individuals, and authenticated by other individuals. In the real world, you have friends who can vouch that you are who you say you are. If you walk into a room full of strangers, your friend can introduce you to the people he knows. Since you trust that your friend is correctly identifying his friends and acquaintances, your trust extends to his friends too.

When you translate the above experience to the electronic, OpenPGP world, it works this way: You create an OpenPGP certificate to identify yourself. When a friend comes to visit, display the certificate. The friend can now sign your certificate (often called “key signing”) and certify that this certificate represents you. Now everyone who trusts the person who signed your key can also trust that your certificate is authentic. A Web of Trust is developed as more people authenticate each certificate. Everyone in the Web of Trust can also exchange messages in the OpenPGP format.

Preparing to use OpenPGP Keys

Once Smartcrypt is installed, the command prompts may assist you in setting up keyring configuration definitions and generating commands to execute Smartcrypt in OpenPGP mode.

Note: `PGPDEF` **configuration settings provide a description of where OpenPGP public and secret keyrings reside. They also provide a name (also referred to as a 'handle') that commands will refer to when requesting OpenPGP key operations for encryption and signature processing.**

Setting Up OpenPGP Keyrings

You can group together collections of OpenPGP keys using named handles. This operation simplifies the process of selecting individual keys for archive recipients and signers.

Note: *PGPDEF* **stores OpenPGP keyrings separately from the Smartcrypt Key Store Index, and does not use the same commands.**

1. In the OpenPGP Administration Page, select ***DEF***.
2. Specify a new/existing dataset in the "Config File:" field
3. When specifying your OpenPGP keyrings for the first time, keep the Edit field set to ***N***. Edit may be used to modify/delete existing PGPKEYRDEFs in the Config File.
4. Enter the required fields in Add PGPKEYRDEF:
   a. **Description**: Description of the Handle
   b. **Handle**: An 8-character name to be referenced with ***-RECIPIENT*** and ***-SIGN_ARCHIVE*** commands.
   c. **Type**: Specify whether the "Key File" is a Public or Private (or Both) key file
   1. **Key File**: File/data set name of the OpenPGP Key File to be assigned to the Handle; one containing private (aka secret) keys for signing and decryption along with their public-key pairings and another only containing public keys of other parties for encryption and signature authentication as described below, usually containing public keys of external trading partners

Configuring Contingency Keys in OpenPGP Mode

You may configure one or more OpenPGP keys to act as contingency keys. A contingency key enables the enterprise to decrypt and access file(s) in an archive when other RECIPIENTs are no longer able or eligible. To define an OpenPGP-based contingency key:

1. Define a read-accessible OpenPGP keyring file (***keyring-file***) containing the public keys to be used as contingency keys.
2. Configure PKCFGSEC to reference the keys to be used as contingency keys. A form of the setting may be chosen such that multiple, or specific keys be included from the keyring.

   Using KEYID:

PKCFGSEC TYPE(*UPDATE) ENTPREC(*PGPDEF 'KEYID=xxxxxxxx' *RQD) PGPKEYPUB(CONTING *FILE '/yourpath/`keyring-file`')

Using CN:
PKCFGSEC TYPE(*UPDATE) ENTPREC(*PGPDEF 'CN=name' *RQD) PGPKEYPUB(CONTING *FILE '/yourpath/`keyring-file`')

Using EM:
**PKCFGSEC TYPE(*UPDATE) ENTPREC(*PGPDEF 'EM=email address' *RQD)
PGPKEYPUB(CONTING *FILE '/yourpath/`keyring-file`')**

Configuration Settings Unique to OpenPGP Processing

PGPDEF

Various operational settings that govern OpenPGP work flow.

"Allow Keys for Smartcrypt" option for PKCFGSEC – Enable OpenPGP keys for ZIP mode processing. This command is NOT used when processing OpenPGP files.

AUTH2PASSALGS(MD5,SHA1,SHA256,SHA384,SHA512)} – When authenticating older OpenPGP files not containing 1-pass signature algorithm information packets, this setting predisposes which hashing algorithms to run for an AUTHCHK. Algorithms MD5 and SHA1 are chosen as defaults because older OpenPGP file implementations were more likely to use these algorithms.

- Additional algorithms may be added to the configuration list if needed.
- An algorithm may be removed from the list to improve performance if no 2-pass authentication is expected for that algorithm.

AUTH2PASSMODE(LITERAL|[BINARY[,]TEXT])} – When authenticating older OpenPGP files not containing 1-pass signature algorithm information packets, this setting predisposes which hashing mode to run for an AUTHCHK. RFC 4880 section 5.2.1 describes the modes of hashing (binary or canonical text documents) declared in the Signature Type tags.

- Default processing will derive the default hashing mode based on the Literal Tag Format field (b-***BINARY*** or t-***TEXT***). This assumes that the creating program of the OpenPGP file synchronized the Literal Data format with that of the Signature mode.
- If a different mode is required (as indicated by the Signature Tag) for authentication to be performed, BINARY, TEXT or both may be set.
- Canonical Text document signature processing in RFC 4880 requires that records be formatted internally with CRLF (x'0D0A') for the purpose of hash calculations, regardless of the actual record delimiters used in the data stream. This introduces processing overhead due to the dual processing of records.
- This setting is not active when a signature algorithm/mode pre-declarative packet is present in the OpenPGP file.

PGPDEF

Defines an OpenPGP Keyring

- One or more PGPDEF statements may be included for the PKPGPZ or PKPGPU command, or implicitly included command streams as directed by the PKCFGSEC module.
- OpenPGP keyrings may be separated by type (for example, one containing private (aka secret) keys for signing and decryption along with their public-key pairings, and another only containing public keys of other parties for encryption and signature authentication), or you can provide a consolidated OpenPGP keyring by declaring the same handle name for public and private ring file names.
- PUB or PVT declares the basic key type for the ring as described below
    - o PUB – Equivalent to an Address Book, usually containing public keys of external trading partners.
        - May be used for encryption
        - May be used for signature authentication
        - Multiple address book specifications may be provided
        - Only public keys are examined (or processed) from this ring
    - o PVT – Equivalent to “My” certificates.
        - May be used in encryption (public key portions only)
        - Used in decryption (passphrase required to access private key in ***ENTPREC*** command)
        - Used in signature creation with ***SIGNERS*** (passphrase required to access private key)
- ***Ring_handle*** declares a unique identifier to be referenced when ***RECIPIENT***s or ***SIGN***ers are requested. (Also useful in reporting)
    - o The same handle name may be used to join a private and secret ring together into one unit.
    - o Multiple PGPDEF statements with different handles may be declared to access different OpenPGP keyrings in the same execution.
- ***FILE*** declares the name of the file-based OpenPGP keyring (for example, the Integrated File System (IFS) file name).

Example 1

In this example, both the public and private keyrings are joined under the handle name “MYSTORE” so that public keys may be accessed from both in a single ENTPREC command:

```
ENTPREC((*PGPDEF MYSTORE 'key_selection_criteria' *RQD))
PGPDEF((MYSTORE *PVT *FILE '/yourpath/secring.pgp')
       (MYSTORE *PUB *FILE '/yourpath/pubring.pgp'))
```

Example 2

In this example, differing public and private keyrings are declared with different handle names so that public keys may be used from each keyring in different ENTPREC commands. In addition, a signature is applied from the local secret keyring. Various forms of selection criteria (Email address, Common name and OpenPGP KEY ID) are also demonstrated.

```
ENTPREC((*PGPDEF US 'EM=key_selection_criteria' *RQD)
        (*PGPDEF THEM 'CN=key_selection_criteria' *RQD))
SIGNERS(*PGPDEF US 'KEYID=key_selection_criteria' passphrase *RQD)
PGPDEF((US *PVT *FILE '/yourpath/our_secring.pgp')
       (THEM *PUB *FILE '/yourpath/their_pubring.pgp'))
```

Example 3

In this example, public keyrings are declared with different handle names representing various departmental keyrings. All valid keys from each ring are included as recipients.

```
ENTPREC((*PGPDEF MRKTING *N *RQD)
        (*PGPDEF SALES *N *RQD)
        (*PGPDEF FINANCE *N *RQD))
PGPDEF((MRKTING *PUB *FILE '/yourpath/MRKTING_pubring.pgp')
       (SALES *PUB *FILE '/yourpath/SALES_pubring.pgp')
       (FINANCE *PUB *FILE '/yourpath/FINANCE_pubring.pgp'))
```

Also see the following members in QCLSRC for additional examples:

```
PKOPGP01 - Sample OpenPGP encryption/signing verification job
PKOPGP02 - Sample create an encrypted ZIP using OpenPGP keys
```

Note: both jobs use the test OpenPGP keys in the archive TSTOPGPZ.

Creating OpenPGP Archives

Creating an OpenPGP archive is basically the same as creating a ZIP archive. See PKPGPZ "PKWARE OpenPGP ZIP" Command.

You must use the ***PKPGPZ*** to create OpenPGP files.

Note: A common technique used with OpenPGP to process multiple files is to place the files into a TAR and then apply OpenPGP encryption and/or digital signatures to the TAR. If you have more than one file to wrap with an OpenPGP key, a two-step process is required. You will first need to create a TAR archive to store the files in. Smartcrypt will then apply OpenPGP to your TAR.

Viewing OpenPGP Files

The steps required to view the contents of an OpenPGP file are essentially the same as with viewing any other supported file type. Use "*VIEW" for the "Type of processing action (TYPE)".

You must use the ***PKPGPU*** to view OpenPGP files.

Opening OpenPGP Files

Opening (or decompressing) an OpenPGP file is basically the same as opening a ZIP, with some modifications to the available options. Use "*EXTRACT" for the "Type of processing action (TYPE)".

Some processing restrictions apply based on the OpenPGP file architecture. For example, the directory information is not independently accessible from the data portion of the OpenPGP file:

- If encrypted, a decryption key is required to perform a View of the filename and related metadata stored in the OpenPGP Literal Tag
- Decryption and Inflation are required to access the filename and other metadata.
- Inasmuch as the OpenPGP file format does not retain file data sizes (compressed or uncompressed), in order for a VIEW request to report on file size information, the entire data file must be decrypted and decompressed for byte counts to be accumulated.

Working with OpenPGP Files Encoded with "ASCII Armor"

Section 2.4 of the OpenPGP standard, RFC 4880, describes ASCII Armor (Radix-64) this way:

> OpenPGP's underlying native representation for encrypted messages, signature certificates, and keys is a stream of arbitrary octets. Some systems only permit the use of blocks consisting of seven-bit, printable text. For transporting OpenPGP's native raw binary octets through channels that are not safe to raw binary data, a printable encoding of these binary octets is needed. OpenPGP provides the service of converting the raw 8-bit binary octet stream to a stream of printable ASCII characters, called Radix-64 encoding or ASCII Armor.

PKPGPU will automatically read any Armor files and Armor key rings, but in some cases there may be a need to convert the Armor file to a binary file. Use PKARMOR to decode an Armor file to a binary file.

When decoding a Radix-64 file, the file is first examined to determine if the file is in ASCII or EBCDIC and whether the file contains End-Of-Line control characters. The file must contain the appropriate OpenPGP header and trailer along with a valid CRC24. The data must all be in BASE64 characters otherwise the decoding will fail.

Example:

PKARMOR TYPE(*DECODE)
INF('/yourpath/armor_archive.pgp')
OUTF('/yourpath/archive.pgp')

Note: All special header keys (Version, Comment, MessageID, Hash, and Charset) are ignored and bypassed.

Note: The utility will only decode one file per ARMOR file. If there are multiple headers inside a file only the first one is decoded.

Valid PGP Header -----BEGIN PGP MESSAGE-----

Valid PGP Trailer: -----END PGP MESSAGE-----

Run the PKSCNPGP utility to verify the OpenPGP binary file.

PKARMOR will also encode an OpenPGP Binary file to an ARMOR file as either ASCII or EBCDIC.

When encoding an OpenPGP to an ARMOR file, each data line of a newly created file will be a maximum of 76 bytes. If the file is to be created in ASCII, the file format is not as important since it will be a stream file with Line Feed characters separating the data lines. A valid OpenPGP header and trailer will be part of the encoding along with a proper CRC24 as per the specifications of RFC 4880.

Example:

PKARMOR TYPE(*ENCODE) MODE(*ASCII)
INF('/yourpath/archive.pgp')
OUTF('/yourpath/armor_archive.pgp')

PKARMOR Command Summary with Parameter Keyword Format

The PKARMOR utility may be used to perform the following:

- DECODE an ASCII-ARMOR (Radix-64) file to a binary file
- ENCODE a file to an ASCII-ARMOR file (ASCII-translation)
- ENCODE a file to an ASCII-ARMOR file (EBCDIC-translation)

Note: Only the first matching OpenPGP message or key header will be processed.

```
                      PK ARMOR Conversion Utility (PKARMOR)
Type choices, press Enter.

Conversion Type  . . . . . . . .                 *DECODE, *ENCODE
  [ENCODE] File Format . . . . .   *EBCDIC       *EBCDIC, *ASCII
  [DECODE] Only Header Type  . .   *MESSAGE      *MESSAGE, *PUBLIC, *PRIVATE
Input File . . . . . . . . . . .


Output File  . . . . . . . . . .
```

PKARMOR Command Keyword Details

[DECODE] Only Header Type(DHDR)

As previously noted, only the first matching OpenPGP message or key header will be processed. If an OpenPGP file contains multiple header types, a specific type may be processed by using one of the following options:

***MESSAGE**	DECODE or ENCODE the first OpenPGP MESSAGE in a file.
***PUBLIC**	DECODE or ENCODE the first OpenPGP PUBLIC key in a file.
***PRIVATE**	DECODE or ENCODE the first OpenPGP PRIVATE key in a file.

Input File(INF)

Specifies the existing IFS path/file name of the input file.

[ENCODE] File Formats(MODE)

The possible file types are:

***EBCDIC**	Output file (OUTF) is to be created as EBCDIC.
***ASCII**	Output file (OUTF) is to be created as ASCII.

Output File(OUTF)

Specifies the IFS path/file name of the output file.

Note: If the output file already exists, it will be overwritten.

Conversion Type(TYPE)

The possible conversion types are:

*DECODE

When decoding a Radix-64 file, the file is first examined to determine if the file is in ASCII or EBCDIC and whether the file contains End-Of-Line control characters. The file must contain the appropriate OpenPGP header and trailer along with a valid CRC24. The entirety of the data must be in BASE64 characters otherwise the decoding will fail.

Note: All special header keys (Version, Comment, MessageID, Hash, and Charset) are ignored and bypassed.

The utility will only decode one message or key per file. If there are multiple headers inside a file, only the first one is decoded.

Example of a valid OpenPGP Header and Trailer:

-----BEGIN PGP MESSAGE-----
-----END PGP MESSAGE-----

Run the CPGPSCAN utility afterward to verify the OpenPGP binary file.

***ENCODE**

Consider your target environment before encoding with PKARMOR. For example, if the intended target of the ASCII-ARMOR file is Windows, UNIX or any other ASCII-based platform, ASCII translation is preferred. If the desired platform is EBCDIC-based, the encoded ASCII-ARMOR file should preferably be defined as Fixed Block with a record length of 80. Each data line of the newly created file will have a maximum of 76 bytes. An ASCII-ARMOR file encoded with ASCII translation consists of a stream file with Line Feed characters separating the data lines. A valid PGP header and trailer will be part of the encoding along with a proper CRC24 as per the specifications of RFC 4880.

OpenPGP Support Exclusions

The following features have been determined to be out of scope for SecureZIP for IBM i v14.

- "Optional" automatic handling of embedded TAR internal files.
- Original file metadata is not preserved in OpenPGP files.
- "Optional" TAR – Without TAR, one and only one file may be processed (although the RFC does not prohibit stacked OpenPGP packets to represent multiple files).
- PKPGPZ is limited to ADD for creating a new OpenPGP file (no updating).
- PKPGPZ file selection is limited to 1 and only 1 file.
- ADVCRYPT(RC4) and ADVCRYPT(AE_2) are not supported for OpenPGP files.
- ADVCRYPT(STANDARD) is not supported for OpenPGP files.
- ADVCRYPT(DES) is not supported for OpenPGP files.
- ADVCRYPT(CAST5) is not supported for ZIP archives.
- FNE(*NO) is not supported (FNE is implied) with OpenPGP files.
- UTF-8 data formats will be detected and handled as BINARY with a message.
- UTF-8 data translation must be prepared by the user before 'literal packet' create.
- SIGNERS to support one signatory for OpenPGP file creation (although spec supports multiple).
- DSA keys are only to be used with signing/authentication for OpenPGP files, not ZIP format archive.

- DSA-1024 and DSA-2048 bit keys only are supported for signing or authentication-TAMPERCHECK.
- SIGNERS(*FILE) is not supported with OpenPGP files.
- zlib and Bzip2 compression methods are supported for the extraction of input OpenPGP archives from other sources.
- AUTHCHK(*FILE) is not supported with OpenPGP files.
- Certificate Revocation Lists (CRLs) are not supported.
- OpenPGP Contingency Keys must be provided in PKCFGSEC.
- ENCRYPT_CERT_LIMIT limits (3275) currently in effect for creation.
- FACILITY(IBMSW) is not supported for CAST5.
- Self extractors do not support OpenPGP extraction.
- FTRAN will be ignored for filename characters (UTF-8 required).
- Only one file (native LITERAL packet, or tar) will be supported for EXTRACTION.
- Non-ZIP conforming OpenPGP encryption key sizes will negate use of keys for ZIP processing.
- Non-ZIP conforming OpenPGP signing key sizes will negate use of keys for ZIP processing.

Signed Message Files

Some OpenPGP products have the capabilities to create “Signed Message” or “Cleartext Signature” files. Section 7 of the OpenPGP standard, RFC 4880, describes the Cleartext Signature Framework. Smartcrypt has the capability to read and process this type of file.

This file type has clear text in ASCII that is readable (non armoring) and the readable text is signed with a valid OpenPGP signature stored in ASCII Armor (Radix-64). The signed text is still readable without special software.

In order to bind a signature to a cleartext, the framework used consists of:

- The cleartext header '-----BEGIN PGP SIGNED MESSAGE-----' on a single line,
- "Hash" Armor Header to define which hash is used for signing,
- Exactly one empty line not included into the message digest,
- The dash-escaped cleartext that is included into the message digest,
- The ASCII armored signature(s) including the '-----BEGIN PGP SIGNATURE-----' Armor Header and '-----END PGP SIGNATURE-----' Armor Tail Lines.

As with binary signatures on text documents, a cleartext signature is calculated on the text using canonical <CR><LF> line endings. The line ending (i.e., the <CR><LF>) before the '-----BEGIN PGP SIGNATURE-----' line that terminates the signed text is not considered part of the signed text. Also, any trailing whitespace -- spaces (0x20) and tabs (0x09) -- at the end of any line is removed when the cleartext signature is generated.

Since the signature is very specific to line control characters and exact text, it is important that the text is not disturbed. You should always transfer this type of file as a binary stream file so it is read as an ASCII file. Since Smartcrypt can handle the file being in EBCDIC, the file should be in variable block. If checking for authentication with AUTHCHK and the message “ZPEX146W PGP: Signature authentication failed” appears on the EBCDIC file, it is highly probable that a transport issue has occurred.

When a hash header is found with verbose, the message “ZPGP014I Signed Message Data will be hashed with Hash: SHA256 Code=0x08” will be shown.

Examining OpenPGP File Structure with PKSCNPGP

The PKSCNPGP utility program is useful in examining the external packet structure of an OpenPGP file. A detail line describing each packet block found is listed with a short description of the type found, the accumulated count of that type, its length (L=) and whether it is the final packet of that type (0-False, 1-True). Note that the size of all intermediate packets of a streamed packet sequence must be a power of 2.

When an OpenPGP file is encrypted, the Encryption Key descriptors (types 1 and/or 3) will be followed by Protected Data packets (either 9 or 18 depending on the mode of encryption chosen by the creating OpenPGP application). All other packets, Compression (8), Literal data (11), Signatures (2 or 4) will be obfuscated within the encrypted data packet stream.

Scan an OpenPGP File with PKSCNPGP

An OpenPGP file may be scanned to report on the outer tag layers (ref. RFC 4880 Section 4.3 “Packet Tags”). When a file is compressed and/or encrypted, underlying tag layers (e.g. Literal and Signatures) will not be visible in the outer layer. However, the type of encryption keys–symmetric key (passphrase) or public key (recipient)–will be evident at the beginning of the file.

Example:

```
CALL      PGM(PKSCNPGP) PARM('V' ‘/yourpath/archive.pgp)
```

Uncompressed/Unencrypted (Literal-only Packet 11) Report

In this sample file, the data was not compressed, encrypted or signed.

```
PKWARE z/OS PGP File Scan 01 - Copyright. 1989-2016 by PKWARE, Inc.
Valid Types only
In file: /yourpath/archive.pgp
 size  :   0
First 256 Bytes <CBEC7427535550504F52542E5454363734372E444154414F55542E46...
02020202020202020202020202020202020202020202024869676820...
92020202020202020202022850544620554B36303831332920202050...
02020202020202020202020202020202020202020202020202020220...
0>
 Packet 11--Literal Data # 1 L=4096 Final=0
      <0x74> Filename(39)<archive.pgp>
 Packet 11--Literal Data # 2 L=4096 Final=0
 Packet 11--Literal Data # 3 L=4096 Final=0
 Packet 11--Literal Data # 4 L=4096 Final=0
. . .
Packet 11--Literal Data #67 L=3936 Final=1
```

```
PKSCNPGP Total found Packet count=67 Len=274341
  67 Total-11--Literal Data
PKSCNPGP - Last  512 Bytes <202020202020202020202020202020202020202...
2020202020202020202020202020202020202020202020343639302B23404C4239202...
2020202020202020202020202020202020202020202020202020202020202020203...
20202020202020202020202020202034363933202A20436F6E646974696F6E616C20415...
202020202020202020202020202020204D533037323430370D0A2020202020202020202...
2020202020202020202049462020202028544D2C4D5456545F454E5649524F4E5F464...
31313930380D0A2030303033343452039313130204130363020202020202020203030303...
204D5456545F454E5649524F4E5F464C4147532C464C475F454E565F41504943414...
PKSCNPGP - buffers read = 274341 <0000000000042FA5>
PKSCNPGP Ending Return Code=0.
```

Compressed, Encrypted (both recipient and passphrase) and Signed Report

In this sample file, encryption was performed for multiple public-key recipients (Packet 1 for each) as well as with a passphrase (Packet 3).

Although the file had also been compressed and signed, those packet tags, along with the Literal tag are encrypted within the encryption stream (Packet 18).

```
PKWARE z/OS PGP File Scan 01 - Copyright. 1989-2016 by PKWARE, Inc.
Valid Types only
In file: /yourpath/archive.pgp
 size  :   0
. . .
 Packet  1--Public-Key Encrypted Session Key # 1 L=268 Final=1
             Key ID=75123FA08390D29D Algo(08)AES192
 Packet  1--Public-Key Encrypted Session Key # 2 L=268 Final=1
             Key ID=831832A5CAE42D9B Algo(08)AES192
 Packet  1--Public-Key Encrypted Session Key # 3 L=268 Final=1
             Key ID=D267C464064512A1 Algo(07)AES128
 Packet  1--Public-Key Encrypted Session Key # 4 L=268 Final=1
             Key ID=C8D383E996511F03 Algo(07)AES128
 Packet  1--Public-Key Encrypted Session Key # 5 L=396 Final=1
             Key ID=9CE8C7D1F7DB405D Algo(0B)Unknown
 Packet  1--Public-Key Encrypted Session Key # 6 L=268 Final=1
             Key ID=3CFD1698E2127C44 Algo(07)AES128
 Packet  3--Symmetric-Key Encrypted Session Key # 7 L= 30 Final=1
             Algo(03)
 Packet 18--Sym. Encrypted & Integrity Protected Data # 8 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data # 9 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #10 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #11 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #12 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #13 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #14 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #15 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #16 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #17 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #18 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #19 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #20 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #21 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data #22 L=1959 Final=1

PKSCNPGP Total found Packet count=22 Len=61106
   6 Total- 1--Public-Key Encrypted Session Key
   1 Total- 3--Symmetric-Key Encrypted Session Key
  15 Total-18--Sym. Encrypted & Integrity Protected Data
. . .
PKSCNPGP - buffers read = 61106 <000000000000EEB2>
PKSCNPGP Ending Return Code=0.
```

Scan an OpenPGP Keyring with PKQRYCDB

Public Keyring Report (ref. “gpg –k”)

PKQRYCDB RUNTYPE(*ALL) FTYPE(*PGPKRF)
CTYPE(*PUBLIC) FNAME('/yourpath/pubring.pgp')
LOGLVL(*MAXLOG)

```
PKSCANCRT 002I File to be OpenPGP KeyRing
PKSCANCRT 005I scan(20) file is: '/yourpath/keyring.pgp'
PKSCANCRT 008I OpenPGP KeyRing #0 found (1186) '/yourpath/keyring.pgp'' Type=20
--- OpenPGP Key 0 ---
PKWAREIVPOPGP1
OpenPGP Key ID: 2D55FE62384F37E4
Subject:
   CN=PKWAREIVPOPGP1
   E=noreply@pkware.com
Issuer:
   CN=PKWAREIVPOPGP1
   E=noreply@pkware.com
SerialNumber:
   D78737CD 2665334E FC96592A 7A1A08D8 F159EA04
NotBefore   (UTC):
   Wed Oct 19 14:14:13 2011
NotAfter    (UTC):
   Mon Jan 18 23:14:07 2038
KeyUsage: 80 00
        :Digital Signature Key Usage
Public Key Hash:
   29 F0 F3 00 D1 10 B8 30 27 D5 88 79 3F E2 00 ED 6D 4C 7D 9C
RSA Public Key - 2048 bits
--- OpenPGP Key 1 ---
PKWAREIVPOPGP1
OpenPGP Key ID: F966907A260233C0
Subject:
   CN=PKWAREIVPOPGP1
   E=noreply@pkware.com
Issuer:
   CN=PKWAREIVPOPGP1
   E=noreply@pkware.com
SerialNumber:
   F579B529 A97F7BFA 17E37BFA 7D4B9FDA E4FD9904
NotBefore   (UTC):
   Wed Oct 19 14:14:13 2011
NotAfter    (UTC):
   Mon Jan 18 23:14:07 2038
KeyUsage: 30 00
        :Data Encipherment Key Usage
        :Key Encipherment Key Usage
Public Key Hash:
   89 9B AA 98 74 34 6A 11 FE CE 08 1D 0A 90 69 BD E4 B9 0D EE
RSA Public Key - 2048 bits

PKSCANCRT 099I  Completed rc=0.  2 OpenPGP Keys processed
```

Private Keyring Report (ref. “gpg –K”)

PKQRYCDB RUNTYPE(*ALL) FTYPE(*PGPKRF)
CTYPE(*PRIVATE) FNAME('/yourpath/secring.pgp')
PASSWORD(password) LOGLVL(*MAXLOG)

```
PKSCANCRT 002I File to be OpenPGP KeyRing
PKSCANCRT 005I scan(20) file is: '/yourpath/keyring.pgp'
PKSCANCRT 008I OpenPGP KeyRing #0 found (2564) '/yourpath/secring.pgp' Type=20
--- OpenPGP Key 0 ---
PKWAREIVPOPGP1
OpenPGP Key ID: 2D55FE62384F37E4
Subject:
   CN=PKWAREIVPOPGP1
   E=noreply@pkware.com
Issuer:
   CN=PKWAREIVPOPGP1
   E=noreply@pkware.com
SerialNumber:
   D78737CD 2665334E FC96592A 7A1A08D8 F159EA04
NotBefore   (UTC):
   Wed Oct 19 14:14:13 2011
NotAfter    (UTC):
   Mon Jan 18 23:14:07 2038
KeyUsage: 80 00
        :Digital Signature Key Usage
Public Key Hash:
   29 F0 F3 00 D1 10 B8 30 27 D5 88 79 3F E2 00 ED 6D 4C 7D 9C
Private Key Available
RSA Public Key - 2048 bits
--- OpenPGP Key 1 ---
PKWAREIVPOPGP1
OpenPGP Key ID: F966907A260233C0
Subject:
   CN=PKWAREIVPOPGP1
   E=noreply@pkware.com
Issuer:
   CN=PKWAREIVPOPGP1
   E=noreply@pkware.com
SerialNumber:
   F579B529 A97F7BFA 17E37BFA 7D4B9FDA E4FD9904
NotBefore   (UTC):
   Wed Oct 19 14:14:13 2011
NotAfter    (UTC):
   Mon Jan 18 23:14:07 2038
KeyUsage: 30 00
        :Data Encipherment Key Usage
        :Key Encipherment Key Usage
Public Key Hash:
   89 9B AA 98 74 34 6A 11 FE CE 08 1D 0A 90 69 BD E4 B9 0D EE
Private Key Available
RSA Public Key - 2048 bits
PKSCANCRT 099I  Completed rc=0.  2 OpenPGP Keys processed
```

Invalid Keyring Report

```
PKSCANCRT 002I File to be OpenPGP KeyRing
PKSCANCRT 005I scan(20) file is: '/yourpath/invalid_keyring.pgp'
PKSCANCRT 008I OpenPGP KeyRing #0 found (800) '/yourpath/invalid_keyring.pgp'
Type=20

PKSCANCRT 099I  Completed rc=8.  0 OpenPGP Keys processed
```

14 PKWARE PartnerLink: SecureZIP Partner

This chapter applies only to participants in the PKWARE SecureZIP Partner program. Other readers may skip this section.

PKWARE SecureZIP Partner enables a *sponsor* organization to give *partner* organizations that may not have ***Smartcrypt for IBM i*** the SecureZIP Partner application so that sponsor and partner can use ***Smartcrypt for IBM i*** to securely exchange ZIP archives.

About *SecureZIP Partner for IBM i*

SecureZIP Partner for IBM i is a special version of ***Smartcrypt for IBM i***. It provides most of the functionality of the full program but works only with archives created by (or for) a sponsor.

SecureZIP Partner has two modes of operation:

- **Read mode**: Read mode enables Smartcrypt functionality to extract files from a ZIP archive signed by a sponsor. In this mode, the program can decrypt and decompress files and authenticate digital signatures.

  In Read mode, the program only extracts; it does not add files to a new or existing archive and does not compress, encrypt, or sign files. SecureZIP Partner extracts only archives digitally signed by a sponsor.

- **Write mode**: Write mode enables Smartcrypt functionality for adding files to a ZIP archive, including commands to compress, encrypt, and digitally sign files.

  In Write mode, the program can create and update archives, but only for a designated SecureZIP Partner sponsor and only if the sponsor provides certificates for SecureZIP Partner to use to encrypt. New or updated archives are automatically encrypted for sponsor recipients: only those recipients can decrypt and read the files.

  SecureZIP Partner only does certificate-based encryption. It does not do passphrase-based encryption.

A single copy of the SecureZIP Partner software can process ZIP archives from multiple sponsors.

See the chapter relating to SecureZIP Partner in the *Smartcrypt for IBM i System Administrator's Guide* for a description of administration and configuration activities unique to the SecureZIP Partner product.

If You Are a Sponsor: Sign the Central Directory

A sponsor organization uses Smartcrypt as usual to work with archives for, or from, a partner. There is just one special requirement when creating an archive for a partner: In order for the partner to be able to extract the archive, you must sign the central directory of the archive using a certificate included in the Sponsor Distribution Package. A Sponsor Distribution Package is a package that PKWARE assembles for a sponsor to configure partners of that sponsor.

Terms and Acronyms Used in This Chapter

The PKWARE SecureZIP Partner program introduces some new concepts and terminology:

- **Sponsor** – An installation responsible for initiating and defining a SecureZIP Partner sponsor-partner relationship with one or more other installations. A sponsor uses the full-featured Smartcrypt product; a partner uses the special ***SecureZIP Partner for IBM i*** version.
- **Partner** – An installation configured using a particular sponsor's Sponsor Distribution Package (see below) to be a partner of that sponsor. A partner uses ***SecureZIP Partner for IBM i*** to work with archives from, or for, the sponsor.
- **Sponsor Distribution Package** – A configuration package distributed to a partner on behalf of a sponsor to define the authorization requirements and provide the certificates needed to process ZIP archives from, or for, the sponsor. The package is digitally signed using a PKWARE-assigned certificate.
- **Sponsor File** – A component file in a Sponsor Distribution Package
- **Sponsor Imprint** – A unique digital representation of a registered sponsor-partner relationship within the PKWARE SecureZIP Partner program. This may represent the unique identification of Distribution Package components or of ZIP archives being read.
- **Sponsor/Partner Registration ID** – A unique registration number that identifies a particular sponsor-partner relationship
- **Read mode** – The mode of ***SecureZIP Partner*** UNZIP processing that extracts archives from (and only from) a SecureZIP Partner sponsor configured on the partner's system
- **Write mode** – The mode of ***SecureZIP Partner*** ZIP processing that creates an encrypted ZIP archive for a particular configured SecureZIP Partner sponsor
- **FF** – Acronym for *full-featured* Smartcrypt operations, as distinct from those of SecureZIP Partner

PKWARE SecureZIP Partner Program: Overview

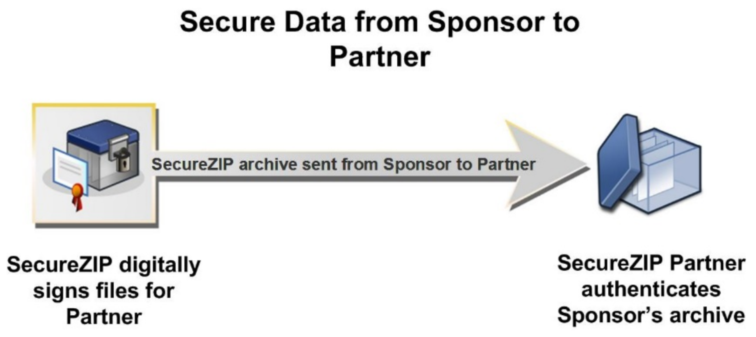
The PKWARE SecureZIP Partner program provides a straightforward, secure way for an organization to exchange sensitive information with outside partners.

A SecureZIP Partner *sponsor* organization establishes a SecureZIP Partner *partner* relationship with another organization. As a SecureZIP Partner partner, the external organization receives the ***SecureZIP Partner for IBM i*** application to use to decrypt and extract archives created by the sponsor using the full Smartcrypt program. The partner can also use the program to create archives for the sponsor that only the sponsor can decrypt.

The SecureZIP Partner program used by a SecureZIP Partner partner extracts archives only *from a sponsor* and creates and encrypts archives only *for a sponsor*.

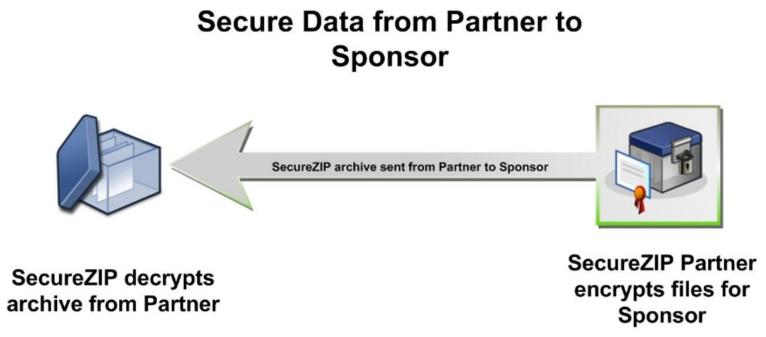
Decrypting and Extracting Sponsor Data (Read Mode)

When SecureZIP Partner is installed at a partner location, a sponsor can create, digitally sign, and encrypt Smartcrypt secure containers (ZIP archives) for the partner. In Read mode, the SecureZIP Partner program verifies that the data file received has the appropriate signature from the sponsor and that the signature is valid. This confirms that the data is from the expected sender and that no tampering has occurred. The partner can then decrypt and extract the data.


Secure Data from Sponsor to Partner
SecureZIP archive sent from Sponsor to Partner
SecureZIP digitally signs files for Partner
SecureZIP Partner authenticates Sponsor's archive

Creating an Archive for a Sponsor

If a sponsor has provided an encryption key, a partner can also use SecureZIP Partner (Write mode) to create encrypted ZIP archives for the sponsor. SecureZIP Partner automatically encrypts any data placed in an archive. The archive can then be transferred to media or transmitted to the sponsor electronically.


Secure Data from Partner to Sponsor
SecureZIP archive sent from Partner to Sponsor
SecureZIP decrypts archive from Partner
SecureZIP Partner encrypts files for Sponsor

Requirements

License

A license key is provided with the installation package for the system administrator to use to activate the ***SecureZIP Partner for IBM i*** product.

Operating Environment

SecureZIP Partner for IBM i requires the same operating environment as full-featured ***Smartcrypt for IBM i***.

Configuring as a Partner for a Sponsor

In order to fully process ZIP Archives, the system administrator for ***SecureZIP Partner for IBM i*** must install one or more Sponsor Distribution Packages and provide the corresponding run-time configuration information for the ZIP and UNZIP jobs to use. The installed Sponsor Distribution Package determines which archive signatures are approved for Read mode extract processing and defines the list of sponsor recipients for whom SecureZIP Partner encrypts new archives.

Functional Overview

SecureZIP Partner for IBM i allows a SecureZIP Partner partner to exchange ZIP archives with a sponsor. A Sponsor Distribution Package provides the partner installation with qualifying controls for processing ZIP archives received from or created for a sponsor. Multiple sponsor profiles with unique processing requirements can be configured to support exchanges with multiple PKWARE SecureZIP Partner sponsors.

A given sponsor profile defines the UNZIP and ZIP capabilities for a partner. In a given sponsor-partner relationship, a partner operates in Read mode to extract archives and in write mode to create archives (if write mode functionality is licensed).

See the *Smartcrypt for IBM i System Administrator's Guide* for information on installing Sponsor Distribution Packages.

General Restrictions

Although many features of full-featured ***Smartcrypt for IBM i*** are also available to ***SecureZIP Partner for IBM i***, some limitations apply for these products.

- ***SecureZIP Partner for IBM i*** (Read mode) can only open a ZIP archive that has been digitally signed by a qualified and configured sponsor, as specified in the Sponsor Distribution Package.
- ***SecureZIP Partner for IBM i*** (Write mode) can only encrypt a ZIP archive for a sponsor-designated set of certificate-based recipients.

Attempts to use features that require operational characteristics outside of the bounds set above are rejected or ignored.

SecureZIP Partner IVP Examples

In the distributed Smartcrypt library, there is a CL program named PLIVPZIP that runs an initial test with the test distributed package from PKWARE with a Sponsor Id number of 0. The following two examples excerpt steps from the CLP.

Read mode example: Step EXTRACT will read in the signed archive by sponsor 0 and will extract the files to a file TMPTEST.   To authenticate the signed archive, AUTHCHK((*ARCHIVE *SPONSOR 0)) is required to read in the sponsor ID number “0” sponsor authentication file.

```
PKUNZIP ARCHIVE('PKW14053L/PLIVPZIP(PLIVPZIP)')
TYPE(*EXTRACT) EXDIR('PKW14053L/TMPTEST')
DROPPATH(*ALL) PASSWORD('PKWARE, Inc.')
AUTHCHK((*ARCHIVE *SPONSOR 0))
```

Sample Results of Step EXTRACT:

```
Digital Certificate Request List:Archive Authenticator
Rqrd    Pub  *SPONSOR                    - a0000000.p7
Archive Authenticator List-----------1 processed:
UNZIP Archive: PKW14053L/PLIVPZIP(PLIVPZIP)
Archive Comment:"Smartcrypt for zSeries by PKWARE"
Searching Archive PKW14053L/PLIVPZIP(PLIVPZIP)  for files to extract
Archive was signed by "PKWARE SecureZIP Partner TEST Signing Certificate" and
verified
Extracting file SECZIP/READER/README.TXT
Inflating *DB:PKW14053L/TMPTEST(READMETXT) Text
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

Write mode example: Step SLNKZIP will read the file that was extracted above and create a new archive by selecting files TMPTEST(READMETXT) for compression with AES256 encryption. The encryption will use the pubic certificates from the Sponsor ID number “0” recipient file with the parameter ENTPREC((*SPONSOR 0)) or ENTPREC((*SPONSOR ‘R0000000.p7’)).

```
PKZIP ARCHIVE('PKW14053L/PLIVPZIP(NEWTESTZ)')
FILES('PKW14053L/TMPTEST(READMETXT)') ADVCRYPT(AES256)
ENTPREC((*SPONSOR 0))
```

Sample Results of Step SLNKZIP:

```
Scanning files in *DB for match  ...
Digital Certificate Request List:Encryption Recipients
Rqrd    Pub  *SPONSOR                    -
/yourpath/PKWARE/PLstore/Sponsor/RECIP/r
0000000.p7
Encryption Recipients List-----------1 processed:
CN=PKWARE SecureZIP Partner TEST Encryption Certificate
EMail=PKWAREPartnerLinkCA@pkware.com
Found  1 matching files
Compressing PKW14053L/TMPTEST(READMETXT) in TEXT mode
Add  PKW14053.L/TMPTEST/READMETX.T  --  Deflating (69%)  encrypt(AES 256Key)
Smartcrypt Compressed 1 files in Archive PKW14053L/PLIVPZIP(NEWTESTZ)
Smartcrypt Completed Successfully
```

Read Mode (UNZIP) Processing

The following features are provided in Read mode:

- An AUTHCHK(Archive) is automatically performed whenever a ZIP archive is opened, except in the following cases:
- An AUTHCHK(ARCHIVE) is requested manually
- Any form of View action
- A TEST action without any form of AUTHCHK request
- A TAMPERCHECK policy will always be enforced for authentication, regardless of the Smartcrypt configuration policy settings.
- The certificate authority trust chain will automatically be honored from the installed and configured Sponsor Distribution Package during archive authentication even if the trusted root certificate is not installed in the local certificate ROOT store.
- If the sponsor also signed files in an archive with the same certificate used to sign the archive central directory, the same certificate authority trust chain used to authenticate the archive signature is used to authenticate signatures on the files.

Restrictions

The following limitations or special behavior applies when ***Smartcrypt for IBM i*** is run in Read/Write mode:

- Archive types (such as GZIP) that do not support signing the archive central directory are not available
- Unsigned archives are rejected for processing

Archive Authentication Settings

The archive authentication that is automatically performed when a ZIP archive is opened for Read mode extract processing uses one or more Sponsor Authentication Configuration Settings to reference an installed Sponsor Authentication File in the certificate store. This is accomplished by including one or more AUTHCHK ((*ARCHIVE *SPONSOR x) (*ARCHIVE *SPONSOR y)) parameters where x and y are sponsor ID numbers.

- At least one AUTHCHK((*ARCHIVE *SPONSOR x)) command is required to access a ZIP archive for extract processing.
- If more than one Sponsor Authentication Configuration Setting command is provided, then the archive authentication will accept an archive from any of the represented sponsors.

Example: Unzipping and authenticating an archive from sponsor 0:

```
➔ PKUNZIP ARCHIVE('PKW14053L/PLIVPZIP(PLIVPZIP)')
    TYPE(*EXTRACT)
    PASSWORD('PKWARE, Inc.') OVERWRITE(*YES)
    EXDIR('PKW14053L/TMPTEST') DROPPATH(*ALL)
    AUTHCHK((*ARCHIVE *SPONSOR 0) )
```

Sample Results:

```
Digital Certificate Request List:Archive Authenticator
Rqrd    Pub  *SPONSOR                   - a0000000.p7
Archive Authenticator List-----------1 processed:
UNZIP Archive: PKW14053L/PLIVPZIP(PLIVPZIP)
Archive Comment:"Smartcrypt for zSeries by PKWARE"
Searching Archive PKW14053L/PLIVPZIP(PLIVPZIP)  for files to extract
Archive was signed by "PKWARE PartnerLink TEST Signing Certificate" and verified
Extracting file SECZIP/READER/README.TXT
Inflating *DB:PKW14053L/TMPTEST(READMETXT) Text
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

Decryption Certificate Selection

RECIPIENT private-key/certificate selection follows the rules for full-featured ***Smartcrypt for IBM i*** local certificate store administration and operations.

File Signature Authentication Certificate Selection

In addition to supporting AUTHCHK *FILES with implicit reference to the AUTHCHK *ARCHIVE certificate validation, separate and distinct file signatory validation can be performed outside of the configured Sponsor Distribution Package. However, this operation is allowed only for files in a sponsor-provided data archive that have signatures for which certificates are not included in the Sponsor Distribution Package.

Public-key certificate files supporting file signature authentication can be supplied through the full-featured ***Smartcrypt for IBM i*** CER certificate types in the local certificate store.

Write Mode (ZIP) Processing

- With Write mode, a sponsor-authorized partner can generate a ZIP archive for the sponsor. Data files placed in the created archive are encrypted for a sponsor-designated set of certificate-based recipients. The following special features are provided by Write mode:
- Unless otherwise specified, a minimum encryption method of AES128 is set for newly encrypted files.
- All recipients defined in the sponsor-defined recipient package (as configured from the Sponsor Distribution Package) are included in the encryption request.
- Recipients identified in the sponsor-defined recipient package are subject to the Smartcrypt ENCRYPOL policy settings in the certificate store configuration. Individual recipients not passing the designated policy attributes are eliminated from encryption processing.
- The certificate authority trust chain from the installed and configured Sponsor Distribution Package is automatically honored for the designated recipients even if the trusted root certificate is not installed in the local certificate store ROOT. A trusted root is included in the Sponsor Distribution Package.

- When a sponsor-created ZIP archive is used as input to create a new target archive, the same features in effect for Read mode are activated for the input archive. In particular, a signed archive is validated with AUTHCHK.
- When a sponsor-source ZIP archive is used as input to create a new target archive, files copied from the original archive are retained in their original form.
- Newly created archives may be viewed in accordance with Smartcrypt functionality.

Restrictions

The following features are not available or have limitations for ***SecureZIP Partner for IBM i***:

- GZIP output is not available.
- Self-extracting archives cannot be created.
- An encryption method for supported recipient-based encryption must be used ("Standard" is not supported).
- Passphrase-based encryption for new archives is not available.
- Encryption is only permitted for sponsor-provided keys.
- All archive creation actions require a qualified response recipient configuration as provided by the Sponsor Distribution Package.
- Directory Integration with LDAP access to private-key certificates for decryption and related command settings is not available.
- An archive can be created and encrypted only for recipients associated with a single sponsor: an ENTPREC request must target a configured sponsor, and an archive cannot be created for multiple sponsors. Note, however, that multiple public-key certificates can be included by a given sponsor in one Sponsor Distribution Package. This implementation rules out the use of DB: and LDAP: request formats for the ENTPREC command.
- An output archive with FNE(*YES) can be created in accordance with the qualified sponsor recipient keys. However, because Write mode can create and encrypt archives only for a sponsor, a partner cannot update a file-name-encrypted archive *from* a sponsor *for* the partner.

Encryption Certificate Selection

ENTPREC public-key/certificate selection is predefined by the Sponsor Distribution Package. The ***Smartcrypt for IBM i*** local certificate store is extended to support sponsor-provided encryption keys with a lookup type of *SPONSOR. The Write mode ENTPREC command is limited to access only those public-keys supplied in the SecureZIP Partner Sponsor Distribution Package.

Sponsor encryption is accomplished by including the ENTPREC((*SPONSOR x)) parameters, where x is the sponsor ID number or sponsor recipient file (R000000x).

One ENTPREC((*SPONSOR x)) is required encrypt the files for the sponsor.

Example: Encrypting files into an archive for sponsor 0:

```
➔ PKZIP  ARCHIVE('PKW14053L/PLIVPZIP(NEWTESTZ)')
    FILES(&FILES1)
    ADVCRYPT(AES256)
    ENTPREC((*SPONSOR 0))
```

Sample Results:

```
Scanning files in *DB for match  ...
Digital Certificate Request List:Encryption Recipients
Rqrd    Pub  *SPONSOR         -
/yourpath/PKWARE/PLstore/Sponsor/RECIP/r0000000.p7
Encryption Recipients List-----------1 processed:
CN=PKWARE PartnerLink TEST Encryption Certificate
EMail=PKWAREPartnerLinkCA@pkware.com
Found  1 matching files
Compressing PKW14053L/TMPTEST(READMETXT) in TEXT mode
Add  PKW14053.L/TMPTEST/READMETX.T  --  Deflating (69%)  encrypt(AES 256Key)
Smartcrypt Compressed 1 files in Archive PKW14053L/PLIVPZIP(NEWTESTZ)
Smartcrypt Completed Successfully
```

15 1Step2Tape Archive Tape Processing

With the 1Step2Tape feature, ***Smartcrypt**ⁱ* can create and read archive files directly to and from tape by using a *tape device file* (file type *TAPF). Writing the archive files directly to tape eliminates the need to provide disk space for temporary archive files that must then be copied to tape.

The tape archiving process has three steps:

1. Define the tape attributes in a tape device file (Note two tape devices described below are distributed with products).
2. Specify the tape device file as the ARCHIVE() parameter with the TYPARCHFL(*TAP) option.
3. Code the options for the parameter PKOVRTAPF() or PKOVRTAPI().

To make an IBM i PKZIP tape archive more manageable to view or extract, PKZIP will build a shadow archive directory file containing directory control information. This is a small tape file with the next sequence number after the newly created archive tape file's sequence number. The file will have a standard tape label of "PKZCDFxxxx" where xxxx is the tape file's sequence number. The shadow file allows processing by tape positioning, instead of reading an entire tape archive file to obtain directory control information normally located at the end of an archive. This shadow archive directory file is only valid running IBM i PKUNZIP. The archive file that is created on tape is a complete standard archive which is compatible with all other PKWARE products.

Creating archive files to tape

The PKZIP command parameter PKOVRTAPF has six options that can override the current tape device file when TYPARCHFL(*TAP) is set. The PKOVRTAPF parameter defaults to the current settings of the *TAPF and to create a shadow directory file. The PKOVRTAPF options are:

```
New Archive Tape Overrides:
    Tape Device         . . . . .   *TAPF    _    Tape Device
    Tape File Label     . . . . .   *TAPF         Tape Header
    Tape Sequence Nbr   . . . . .   *TAPF         1-16777216, *TAPF, *END
    File expiration date . . . .    *TAPF         Date, *NONE, *PERM, *TAPF
    End Of Tape Option . . . . .    *TAPF         *TAPF, *REWIND, *UNLOAD...
    Shadow Dir File     . . . . .   *CSDF         *CSDF, *NO
```

Notes and Suggestions for Writing Archives to Tape

- We recommend you always create the shadow directory file for unzip efficiency.
- To reduce run time, pre-initialize the tapes by defining the tape drive, new volume name and the tape density. For example:

➔ **INZTAP DEV(TAP01) NEWVOL(PKZIP1) DENSITY(*QIC525)**

- The tape device file should be tailored for your environment.
- If there are files already on the tape, the open may take much longer depending on the TAPF sequence number (SEQNBR).
- If SEQNBR is set to *END, ***Smartcrypt**ⁱ* adds the new archive as the last file on the tape. If SEQNBR is set to 1, ***Smartcrypt*** ^{i} starts at the beginning and overwrites any files currently on the tape. The checking of current files on a tape is done using the normal IBM file checking based on expiration date, label processing, and so on.
- If you are writing several archives to the same tape, it will run faster if you use the *LEAVE option for the end-of-tape option.
- If you have a tape drive and tape format that supports Optimum Block, make sure to configure the tape device file to utilize it.

Usage Notes:

Archives written directly to tape by ***Smartcrypt*** ^{i} use the ZIP64 data descriptor records format. This ZIP file structure is documented in the ZIP File Format Specification (APPNOTE) published by PKWARE. Not all ZIP-compatible products support data descriptors or ZIP64, and you may experience problems reading these archives if you need to process them for any reason outside of your IBM i environment. We recommend using only PKWARE, Inc. products to ensure the successful recovery of data from your tape archives.

Smartcrypt ^{i} issues a tape override for the tape device file at the activation group scope level. If other overrides were made with OVRTAPF previously or at the job level for DEV(), LABEL(), SEQNBR(), EXPDATE(), and ENDOPT() parameters, then ***Smartcrypt*** ^{i} will not be able to supersede the prior override.

For example, if an earlier override was OVRTAPF FILE(PKTAPEO1) LABEL('My Header'), and the PKZIP PKOVRTAPF contained (*TAPF 'ARCHIVE_TEST24' *END *TAPF *TAPF), then the label written to the tape will be 'My Header'.

Reading archive files from tape:

PKUNZIP tape override commands may be specified prior to or during the PKUNZIP run with the PKOVRTAPI parameter. PKOVRTAPI includes four options to override the current tape device file when TYPARCHFL(*TAP) is set. The PKOVRTAPI parameter defaults to the current settings of the *TAPF. The PKOVRTAPI options are:

```
Input Archive Tape Overrides:
    Tape Device        . . . . .   *TAPF         Tape Device
    Tape File Label    . . . . .   *TAPF
    Tape Sequence Nbr  . . . . .   *TAPF         1-16777216, *TAPF, *NEXT
    End Of Tape Option . . . . .   *TAPF         *TAPF, *REWIND, *UNLOAD...
```

Notes and Suggestions for Reading Archives from Tape

- If the tape contains shadow directory files, always reference these files to avoid reading the whole archive for processing.
- If the archive consists of multiple tape volumes and you have a shadow directory file on the last volume: Mount the last volume and reference the shadow directory file sequence number. The system will then ask to mount the first file to process the actual archive file for extraction.
- If you are reading several archives from the same tape, it will run faster if you use the *LEAVE option.

Usage Notes:

Smartcrypt^i issues a tape override for the tape device file at the activation group scope level. If other overrides were made with OVRTAPF previously or at the job level for DEV(), LABEL(), SEQNBR(), and ENDOPT() parameters, then ***Smartcrypt***^{i} will not be able to supersede the prior override.

For example, if an earlier override was OVRTAPF FILE(PKTAPEI1) LABEL('My Header'), and the PKZIP PKOVRTAPI contained (*TAPF 'ARCHIVE_TEST24' *NEXT *TAPF), then the label read from the tape will be 'My Header'.

During PKUNZIP processing, the tape will be positioned at the end of the archive to extract data about the files contained in the archive. To extract or test the files in the archive, PKUNZIP will close the tape file and reopen the archive file to position itself for the processing of each file that is selected. This initialization process may take some time, so PKUNZIP will issue the message AQZ0556 "Input Archive Tape Repositioning", after directory data collection has been completed. After the tape has been repositioned, PKUNZIP continues to extract the data as usual.

Setting Up or Changing a Tape Device File for PKZIP or PKUNZIP

To write or read an archive directly to/from tape, the parameter for ARCHIVE() should be a tape device file that defines the tape input/output attributes for the archive. Do not confuse tape device files with data files on the tape volumes. For processing volumes which contain data files, the tape device files provide a link between the application program and the tape device.

Before using tape archives with this product, first tailor the tape device files for the environment.

Included as part of the distribution library are the tape device files named PKTAPEO1 for PKZIP output and PKTAPEI1 for PKUNZIP input. You can view the contents by doing a CHGTAPF pkziplib/PKTAPEx1 and pressing the F4 key. You can tailor these files for your environment with the CHGTAPF command, or you can create and use any other TAPF object that you want. For more information, refer to the CRTTAPF, CHGTAPF, OVRTAPF, and DLTTAPF commands in the IBM CL programmers guide, or reference the IBM site for IBM i: http://publib.boulder.ibm.com/eserver/ibmi.html

Output Tape Device File for PKZIP

In the PKTAPEO1 tape device file, the LABEL parameter specifies the data file identifier or tape header label of the archive file on tape. The PKZIP command provides overrides to the DEV, SEQNBR, LABEL, EXPDATE, and the ENDOPT parameters.

The distributed PKWARE TAPF object was created with the following command:

➔ **CRTTAPF FILE(pkziplib/PKTAPEO1) DEV(TAP01) VOL(*NONE) REELS(*SL) SEQNBR(*END) LABEL('PKZIP.ARCHIVE') FILETYPE(*DATA) TEXT('Smartcrypt Archive Tape File') BLKLEN(262112) RCDBLKFMT(*FB) CODE(*EBCDIC) EXPDATE(*PERM) ENDOPT(*REWIND)**

The contents of the distributed tape device file are shown below:

```
                           Change Tape File (CHGTAPF)

File . . . . . . . . . . . . . . . FILE          PKTAPEO1
  Library  . . . . . . . . . . .                    PKZIPLIB
Tape device  . . . . . . . . . . . DEV            TAP01
                          + for more values
Volume identifier  . . . . . . . . VOL            *NONE
                          + for more values
Tape reels specifications:         REELS
  Label processing type  . . . .                  *SL
  Number of reels  . . . . . . .                  1
Sequence number  . . . . . . . . . SEQNBR         *END
Tape label . . . . . . . . . . . . LABEL          'PKZIP.ARCHIVE    '
Text 'description' . . . . . . . . TEXT           'Smartcrypt Archive Tape File'
                           Additional Parameters

Record length  . . . . . . . . . . RCDLEN         *CALC
Block length . . . . . . . . . . . BLKLEN         262112
Buffer offset  . . . . . . . . . . BUFOFSET       0
Record block format  . . . . . . . RCDBLKFMT      *FB
Extend:                            EXTEND
  Extend file  . . . . . . . . .                  *NO
  Check file . . . . . . . . . .
Tape density . . . . . . . . . . . DENSITY        *DEVTYPE
Data compaction  . . . . . . . . . COMPACT        *DEVD
Code . . . . . . . . . . . . . . . CODE           *EBCDIC
Creation date  . . . . . . . . . . CRTDATE        *NONE
File expiration date . . . . . . . EXPDATE        *PERM
End of tape option . . . . . . . . ENDOPT         *REWIND
User label program . . . . . . . . USRLBLPGM      *NONE
  Library  . . . . . . . . . . .
User specified DBCS data . . . . . IGCDTA         *NO
Maximum file wait time . . . . . . WAITFILE       *IMMED
Share open data path . . . . . . . SHARE          *NO
```

Tape Device Requirements for Writing Archives

- The TAPF device file type must be data or FILETYPE(*DATA) or the archive data will become corrupted
- The tape device file record length (RCDLEN) MUST not exceed 32,764 (the default for PKZIP)
- If the tape device file block length (BLKLEN) is changed, it must be changed by a multiple of 32,764 in the CHGTAPF or CRTTAPF command. On the OVRTAPF command, the BLKLEN must be a multiple of the RCDLEN if the RCDLEN is also overridden.

- The record block format must be fixed block or RCDBLKFMT(*FB)
- The record labels processing must be standard labels or REELS(*SL)
- **IMPORTANT**: If your tape drive or tape format does not support optimum blocks then change the BLKLEN() to *CALC.
- Tape Compression/Compaction: When creating an archive to tape in a non-store mode, overall performance will improve by turning off the tape compaction feature. After compression and/or encryption, data is usually so random that it cannot be further compacted, but the system will test each buffer anyway and sometimes will try to compact. Also, most tape system compaction inserts bytes even when not able to compact.

To turn off tape compaction, you can either change the tape device file for COMPACT(*NO) or issue a tape override prior to PKZIP. For example:

- OVRTAPF FILE(PKTAPEO2) COMPACT(*NO)
- Support for Optimum Blocks: If you have a tape drive and tape format that supports Optimum Block, you can define a new tape device file to improve performance.

First determine the maximum optimum block size for your tape device and density. Then set up a tape device file to define the BLKLEN (not to exceed the tape drive maximum size). Define the BLKLEN as a multiple of the default record length of 32,764. For example, if the tape drive model is a SLR60 drive, and the format of the tape is *SLR60, then Optimum Block is supported with a maximum optimum block size of 256K (262,144 bytes). Overall performance improves with a large block size because fewer tape writes are necessary.

The following examples use a block size of 262,112 (a multiple of the 32,764 record size):

➔ **CRTTAPF FILE(pkziplib/PKTAPEO2) DEV(TAPxx) VOL(*NONE)
REELS(*SL) SEQNBR(*END) LABEL('PKZIP.ARCHIVE')
FILETYPE(*DATA)
BLKLEN(262112) RCDBLKFMT(*FB)
TEXT('Smartcrypt Archive Tape File Optimum Block')
CODE(*EBCDIC) EXPDATE(*PERM) ENDOPT(*REWIND)**

- To use a smaller record size than the default of 32,764, you must issue an OVRTAPF command prior to each PKZIP run. Changing the RECLEN in the tape device file does not change record size.

The following example sets the record size to 8192 and the block size to a multiple of this. The block size must be a multiple of the record length.

➔ **OVRTAPF FILE(PKTAPEO2) RCDLEN(8192) BLKLEN(262144)
RCDBLKFMT(*FB)**

Input Tape Device File for PKUNZIP

In the PKTAPEI1 tape device file, the LABEL parameter specifies the data file identifier or tape header label of the archive file on tape. The PKUNZIP command provides overrides to the DEV, SEQNBR, LABEL, and the ENDOPT parameters.

The distributed PKWARE TAPF object was created with the following command:

➔CRTTAPF FILE(pkziplib/PKTAPEI1) DEV(TAP01) VOL(*NONE) REELS(*SL) SEQNBR(*NEXT) LABEL(*NONE) FILETYPE(*DATA) TEXT('Smartcrypt Tape In ArchiveFile') RCDBLKFMT(*FB) CODE(*EBCDIC) EXPDATE(*NONE) ENDOPT(*REWIND))

The contents of the distributed tape device file PKTAPEI1 are:

```
                           Change Tape File (CHGTAPF)
File . . . . . . . . . . . . . . > PKTAPEI1      Name
  Library  . . . . . . . . . . .     *LIBL       Name, *LIBL, *CURLIB
Tape device  . . . . . . . . . .   TAP01         Name, *SAME, *NONE
               + for more values
Volume identifier  . . . . . . .   *NONE         Character value, *SAME, *NONE
               + for more values
Tape reels specifications:
  Label processing type  . . . .   *SL           *SAME, *SL, *NL, *NS, *BLP...
  Number of reels  . . . . . . .   1             1-255, *SAME
Sequence number  . . . . . . . .   *NEXT         1-16777215, *SAME, *END...
Tape label . . . . . . . . . . .   '*NONE            '
Text 'description' . . . . . . .   'Smartcrypt for IBM i *TAPF Object'
Record length  . . . . . . . . .   *CALC         Number, *SAME, *CALC
Block length . . . . . . . . . .   *CALC         1-524288, *SAME, *CALC
Buffer offset  . . . . . . . . .   0             Number, *SAME, *BLKDSC
Record block format  . . . . . .   *FB           *SAME, *FB, *F, *V, *VB...
Extend:
  Extend file  . . . . . . . . .   *NO           *SAME, *NO, *YES
  Check file . . . . . . . . . .                 *NOCHECK, *CHECK
Tape density . . . . . . . . . .   *DEVTYPE      *SAME, *DEVTYPE, *FMT3480...
Data compaction  . . . . . . . .   *DEVD         *SAME, *DEVD, *NO
Code . . . . . . . . . . . . . .   *EBCDIC       *SAME, *EBCDIC, *ASCII
Creation date  . . . . . . . . .   *NONE         Date, *SAME, *NONE
File expiration date . . . . . .   *NONE         Date, *SAME, *NONE, *PERM
End of tape option . . . . . . .   *REWIND       *SAME, *REWIND, *LEAVE...
User label program . . . . . . .   *NONE         Name, *SAME, *NONE
  Library  . . . . . . . . . . .                 Name, *LIBL, *CURLIB
Maximum file wait time . . . . .   *IMMED        Seconds, *SAME, *IMMED, *CLS
Share open data path . . . . . .   *NO           *SAME, *NO, *YES
```

Tape Device Requirements for Reading Archives

- The TAPF device file type must be data or FILETYPE(*DATA) or the archive data will become corrupted
- The tape device file record length (RCDLEN) if used MUST be *CALC.
- The LABEL() should be code *NONE. The label is best controlled with PKOVRTAPI().
- Change the SEQNBR() to *NEXT to avoid always reading file 1.

Sample - Creating an Archive Directly to Tape

The following examples show the steps to create two archives directly to tape along with directory shadow files. Most tape processing will normally be performed with a CL program, but it can also be done interactively, as in these examples. For another CLP sample, refer to member PKSAMP07 in the QCLSAMP source file distributed with ***Smartcrypt^{i}***.

First, make sure the tape has been properly initialized with standard labels. This can be done using the INZTAP command with appropriate parameters for your environment.

➔ **INZTAP DEV(TAP01) NEWVOL(PKZIP1)**

For this example, we want the archive to be the first sequence file on the tape. We want the label to be ARCHIVE_TEST01 and an expiration date of 11/08/2013:

➔ **PKZIP ARCHIVE('PKW14053S/PKTAPEO1') FILES('testlib/myfile') TYPARCHFL(*TAP) PKOVRTAPF(*TAPF 'ARCHIVE_TEST01' 1 '11/08/2013' *TAPF *CSDF)**

```
Archive File System *TAP active
Scanning files in *DB for match  ...
Tape Archive PKW14053S/PKTAPEO1 being created
Compressing TESTLIB/MYFILE(MYMBR) in TEXT mode
Add  TESTLIB/MYFILE/MYMBR  --  Deflating (67%)
Archive <ARCHIVE_TEST01> Seq(1) created on Tape Volume(PKZIP1).
Creating Tape Shadow Archive Directory File.
Tape Shadow Directory File <PKZCDF0002> Seq(2) created on Tape Volume(PKZIP1).
Smartcrypt Compressed 1 files in Archive PKW14053S/PKTAPEO1
Smartcrypt Completed Successfully
```

The next example creates an archive at the end of the tape. We give the header the name ARCHIVE_TEST02. By using the *END command, this archive will be the next sequence number. Since PKZIP created a shadow directory file in the first example, this will end up being tape file sequence number 3. This time we specify VERBOSE(*MAX) to show the overrides processed:

➔ **PKZIP ARCHIVE('PKW14053S/PKTAPEO1') FILES('testlib/myfil*') TYPARCHFL(*TAP) PKOVRTAPF(*TAPF 'ARCHIVE_TEST02' *END *TAPF *REWIND) VERBOSE(*MAX)**

```
Archive File System *TAP active
Scanning files in *DB for match  ...
Archive override <OVRTAPF FILE(PKTAPEO1) TOFILE(PKW14053S/PKTAPEO1)
LABEL('ARCHIVE_TEST02') SEQNBR(*END) ENDOPT(*REWIND) OVRSCOPE(*ACTGRPDFN)>
Include parameters supplied       1
Found  9 matching files
Tape Archive PKW14053S/PKTAPEO1 being created
Compressing TESTLIB/MYFILE(MYMBR) in TEXT mode
Stats:  (in=55271) (out=18146) (Encrypt=0)
Add  TESTLIB/MYFILE/MYMBR  --  Deflating (67%)
Compressing TESTLIB/MYFILEGER(MYMBR) in TEXT mode
Stats:  (in=48) (out=43) (Encrypt=0)
Add  TESTLIB/MYFILEGE.R/MYMBR  --  Deflating (10%)
Compressing TESTLIB/MYFILETEXT(MYMBR) in TEXT mode
.....
.....
.....
Archive <ARCHIVE_TEST02> Seq(3) created on Tape Volume(PKZIP1).
Archive override <OVRTAPF FILE(PKTAPEO1) TOFILE(PKW14053S/PKTAPEO1)
LABEL('PKZCDF0004') SEQNBR(4) ENDOPT(*REWIND) OVRSCOPE(*ACTGRPDFN)>
Creating Tape Shadow Archive Directory File.
Tape Shadow Directory File <PKZCDF0004> Seq(4) created on Tape Volume(PKZIP1).
Smartcrypt Compressed 9 files in Archive PKW14053S/PKTAPEO1
Smartcrypt Completed Successfully
```

The following command lists the files on the tape after several runs of PKZIP.

➔ **DSPTAP DEV(TAP01) OUTPUT(*PRINT)**

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
5722SS1 V5R3M0 040528                  TAPE VOLUME INFORMATION         PKZIP1
 Device . . . . . :   TAP01                 Volume . . . . . :   PKZIP1
 Owner ID . . . . :                         Density  . . . . :   *SLR60
 Type . . . . . . :   *SL                   Code . . . . . . :   *EBCDIC
                              Record
                  File      Block   Recg  Record  Block   File       Date
Data File Label  Sequence   Format  Tech  Length  Length  Length     Created
 Expiration
 Date

 ARCHIVE_TEST01   0000000001 *FB     P     32764   262112  0000000001 08/04/09
11/08/09
 PKZCDF0002       0000000002 *FB     P     32764   262112  0000000001 08/04/09
11/08/09
 ARCHIVE_TEST02   0000000003 *FB     P     32764   262112  0000000001 08/04/09
*NONE
 PKZCDF0004       0000000004 *FB     P     32764   262112  0000000001 08/04/09
*NONE
                        * * * * *   E N D   O F   L I S T I N G  * * * * *
```

Note that the shadow directory file took on the same expiration date as the archive.

Sample - Extracting Files from an Archive Written Directly from Tape

The following samples show the steps to read archives directly from tape utilizing the new PKOVRTAPI parameter.

First sample will test the files in the archive in tape file number 2 (the shadow directory file for archive file sequence # 1) using the PKOVRTAPI parameter to specify only the sequence number 2 with no label checking:

➔ **PKUNZIP ARCHIVE('PKW14053S/PKTAPEI1') TYPARCHFL(*TAP) VERBOSE(*ALL) TYPE(*TEST) PKOVRTAPI(*TAPF *NONE 2 *TAPF)**

```
Archive File System *TAP active
Archive override <OVRTAPF FILE(PKTAPEI1) TOFILE(PKW14053S/PKTAPEI1) LABEL('*NONE')
SEQNBR(2) OVRSCOPE(*ACTGRPDFN)>
Opening Input Tape Archive PKW14053S/PKTAPEI1
Processing Directory for <ARCHIVE_TEST01> Seq(1) Volume(PKZIP1) Archive.
Input Archive Tape Repositioning for Processing of First Local Directory.
Searching Archive PKW14053S/PKTAPEI1  for files to extract
Testing: TESTLIB/MYFILE/MYMBR
TESTLIB/MYFILE/MYMBR tested OK
Archive <ARCHIVE_TEST01> File Seq(1) Processed tape device file
PKW14053S/PKTAPEI1.
SecureUNZIP   Completed Successfully
```

Note the message: Processing Directory for <ARCHIVE_TEST01> Seq(1) Volume(PKZIP1) Archive. This is the result of referencing the shadow directory file to access the archive file.

Second sample will view the contents of the third tape archive file using the PKOVRTAPI parameter to specify both the tape label ARCHIVE_TEST02 and the sequence number 3. This sample is not reading the shadow directory file:

➔ **PKUNZIP ARCHIVE('PKW14053S/PKTAPEI1') TYPARCHFL(*TAP) VERBOSE(*ALL) TYPE(*VIEW) PKOVRTAPI(*TAPF ' ARCHIVE_TEST02' 3 *TAPF)**

```
Archive File System *TAP active
Archive override <OVRTAPF FILE(PKTAPEI1) TOFILE(PKW14053S/PKTAPEI1)
LABEL('ARCHIVE_TEST02') SEQNBR(3) OVRSCOPE(*ACTGRPDFN)>
Opening Input Tape Archive PKW14053S/PKTAPEI1
Archive:  PKW14053S/PKTAPEI1, 0 bytes, 9 files, 1 Segment
  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
    55271 Defl:S    18146  67% 03-05-09 14:48 13abbd41 TESTLIB/MYFILE/MYMBR
       48 Defl:S       43  10% 10-26-05 07:48 1b277b62 TESTLIB/MYFILEGE.R/MYMBR
      259 Defl:S      210  19% 10-26-05 07:48 b5dbf80c TESTLIB/MYFILETE.XT/MYMBR
    55271 Defl:S    18146  67% 08-17-06 11:08 13abbd41 TESTLIB/MYFILE1/MYFILE
    55271 Defl:S    18146  67% 08-23-06 12:19 13abbd41 TESTLIB/MYFILE2/MYFILE1
    55271 Defl:S    18146  67% 08-23-06 12:19 13abbd41 TESTLIB/MYFILE2/MYFILE2
       48 Defl:S       43  10% 10-26-05 07:48 1b277b62 TESTLIB/MYFILE27.3/MYMBR
    55271 Defl:S    18146  67% 08-17-06 11:08 13abbd41 TESTLIB/MYFILE3/MYFILE
    55271 Defl:S    18146  67% 08-17-06 11:08 13abbd41 TESTLIB/MYFILE4/MYFILE
 --------         -------  ----                          -------
  331981           109172  67%                           9 files
Archive <ARCHIVE_TEST02> File Seq(3) Processed tape device file
PKW14053S/PKTAPEI1.
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

Third sample will extract the files in the third tape archive file using the PKOVRTAPI parameter to specify the tape device "TAP01", the sequence number 4 (for the shadow directory file) and the end option of *REWIND:

➔ **PKUNZIP ARCHIVE('PKW14053S/PKTAPEI1') TYPARCHFL(*TAP) VERBOSE(*ALL) TYPE(*EXTRACT) PKOVRTAPI(TAP01 *NONE 4 *REWIND)**

```
Archive File System *TAP active
Archive override <OVRTAPF FILE(PKTAPEI1) TOFILE(PKW11053S/PKTAPEI1) DEV(TAP01)
LABEL('*NONE') SEQNBR(4) ENDOPT(*REWIND) OVRSCOPE(*ACTGRPDFN)>
Opening Input Tape Archive PKW11053S/PKTAPEI1
Processing Directory for <ARCHIVE_TEST02> Seq(3) Volume(PKZIP1) Archive.
Input Archive Tape Repositioning for Processing of First Local Directory.
Searching Archive PKW11053S/PKTAPEI1  for files to extract
Extracting file TESTLIB/MYFILE/MYMBR
Extracting file TESTLIB/MYFILEGE.R/MYMBR
Extracting file TESTLIB/MYFILETE.XT/MYMBR
Extracting file TESTLIB/MYFILE1/MYFILE
Extracting file TESTLIB/MYFILE2/MYFILE1
Extracting file TESTLIB/MYFILE2/MYFILE2
Extracting file TESTLIB/MYFILE27.3/MYMBR
Extracting file TESTLIB/MYFILE3/MYFILE
Extracting file TESTLIB/MYFILE4/MYFILE
Archive <ARCHIVE_TEST02> File Seq(3) Processed tape device file
PKW11053S/PKTAPEI1.
SecureUNZIP  extracted     9 files
Smartcrypt Completed Successfully
```

How to Copy a Tape Archive to a Disk File

If a need arises to copy a tape archive to disk for any reason, we suggest following these steps.

1. Identify the archive. Make sure it is the archive file and not the shadow directory file. If needed use DSPTAP to identify the correct tape file.
2. Create a file in a library where the file from tape will be copied. The record length must be the same as the record length on the tape file.

   ➔ **CRTPF FILE(MYLIB/TEMPZIP) RCDLEN(32764) FILETYPE(*DATA) MBR(*NONE) MAXMBRS(*NOMAX)**

3. Run the CPYFRMTAP command to copy the tape file to disk.

➔ **OVRTAPF FILE(PKTAPEO1) TOFILE(*LIBL/PKTAPEO1) DEV(TAP01) SEQNBR(3) LABEL(ARCHIVE_TEST02)**

➔ **CPYFRMTAP FROMFILE(*LIBL/PKTAPEO1) TOFILE(MYLIB/TEMPZIP) TOMBR(MYNBR1) FROMENDOPT(*REWIND) MBROPT(*ADD)**

```
File PKTAPEO1 overridden to PKTAPEO1 in PKZIPLIB.
RCDLEN, BLKLEN, RCDBLKFMT, BUFOFSET values assumed.
Member or label overridden to ARCHIVE_TEST01.
Member MYNBR1 added to file TEMPZIP in ATEST.
47 records copied from member ARCHIVE_TEST01.
```

4. At this point, the copied disk archive could be processed with PKUNZIP/PKZIP.

➔ **PKUNZIP ARCHIVE('MYLIB/TEMPZIP/MYNBR1') TYPE(*VIEW)**

```
Archive:  MYLIB/TEMPZIP(MYNBR1), 131056 bytes, 9 files, 1 Segment
  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
    55271 Defl:S    18146  67% 03-05-09 14:48 13abbd41 TESTLIB/MYFILE/MYMBR
       48 Defl:S       43  10% 10-26-05 07:48 1b277b62 TESTLIB/MYFILEGE.R/MYMBR
      259 Defl:S      210  19% 10-26-05 07:48 b5dbf80c TESTLIB/MYFILETE.XT/MYMBR
    55271 Defl:S    18146  67% 08-17-06 11:08 13abbd41 TESTLIB/MYFILE1/MYFILE
    55271 Defl:S    18146  67% 08-23-06 12:19 13abbd41 TESTLIB/MYFILE2/MYFILE1
    55271 Defl:S    18146  67% 08-23-06 12:19 13abbd41 TESTLIB/MYFILE2/MYFILE2
       48 Defl:S       43  10% 10-26-05 07:48 1b277b62 TESTLIB/MYFILE27.3/MYMBR
    55271 Defl:S    18146  67% 08-17-06 11:08 13abbd41 TESTLIB/MYFILE3/MYFILE
    55271 Defl:S    18146  67% 08-17-06 11:08 13abbd41 TESTLIB/MYFILE4/MYFILE
 --------         -------  ----                           -------
  331981           109172  67%                            9 files
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

A Performance Considerations

This appendix lists a few performance considerations when running ***Smartcrypt**^{i}*. Most performance related issues can be controlled by the PKZIP/PKUNZIP parameters. However, it should be noted that PKZIP data compression is CPU intensive by its very nature, and that PKZIP/PKUNZIP parameters can only help to a limited degree. Therefore, it should be expected that a *reasonable* amount of CPU resources will be needed for such operations.

Interactive Performance

When compressing large size files, PKZIP will sometimes use as much CPU resources as the system will allow. With this in mind, processing very large files may perform best as a submitted job. However, some IBM i environments have constraints on running interactive jobs. If those interactive jobs run for a long time and use a high amount of CPU resources, the system will slow down and may issue the message CPI1479 "Interactive activity approaching capacity of installed feature." In this case, review the details of this message. This usually means that the interactive systems are using more resources than the IBM i was configured to use.

Compression Type Performance

Selecting a compression method is one way to get the smallest compressed file with the relationship to the CPU usage and run times. Sometimes, to get the best results, you may have to run several tests with the data to balance the compression ratio to the length of the run time. Running with *MAX will usually get the best compression ratio but will also run the longest. In most of our test cases, *MAX would run 30%-40% longer than *NORMAL and might only gain less than 1% better ratio. This is why we recommend using SUPERFAST (the default) unless your testing implies otherwise.

To minimize the overhead needed to ZIP, the best thing (and the easiest) is to select a compression method other than *MAX. PKZIP's default compression method is SUPERFAST.

When using the compression method of Maximum, you are only compressing the data by another 1-8% over a job that might use the SUPERFAST compression method. The archive file size change is minimal. However, the time difference between a maximum and a SUPERFAST job can be measured in hours if the file is big enough!

You may read more about the compression levels by prompting the Compression Level parameter (F1).

```
Compression Level  . . . . . . . *SUPERFAST    *FAST, *NORMAL, *MAX...
```

Data Type Selection

Getting the best performance from your IBM i machine with regards to a PKZIP job can truly depend on the parameters you have selected for the job. In many cases, the compressed size of a file depends on the type of data (Binary vs. Text), and the compression type selected. Text will usually compress more since it has a higher probability of repeated characters.

Knowing the target platform of the data will help you resolve how PKZIP is to treat the data during the compression process. However, PKZIP treatment of data defaults to *DETECT. *DETECT means that PKZIP will scan the data (up to 97% of the input file) to determine whether the data that it is going to compress should be treated as TEXT or BINARY. This can be an especially painful process if you are selecting large files for compression. However, to get around the *scanning* overhead, if you know you are sending the archive or ZIP file to a PC or to a UNIX machine, you know that the data will need to be converted to TEXT (or ASCII). Therefore, you should select file Types(*TEXT). If the data is targeted for another IBM i machine, then you should select *BINARY. *DETECT should only be used when you do not know the nature of the data.

You may read more about the data types by prompting the file Types parameter (F1).

```
File Types . . . . . . . . . . .   *DETECT  *DETECT *TEXT *BINARY ....
```

Archive Placement (IFS or in a Library)

For best performance, try to store the archives in the IFS. By placing the archive in the IFS instead of in a library/file reduces the overall CPU usage and in some cases can reduce the run times as much as 30%-40%.

It is recommended when using the ZIP process for large files that the ZIP archive be stored in the IFS. This method provides the best performance and makes the most efficient use of storage space for both ZIP archives and ZIP temporary files.

ZIP64 Processing Considerations

When processing very large files or high volumes of files, the processing characteristics of PKZIP may vary depending on the phase of processing involved. Some common processing phases and their run-time characteristics are:

- ZIP file selection: When selecting a very large number of files through many directories and/or libraries, the initial selection requires IO time and memory per file to analyze and manage each of the file’s properties. The more files to select, the more memory and initial startup overhead. Each site will have to discover their practical limits based on their environments and resources.

- Archive directory read processing: When updating an existing archive that contains a very large number of files, time and memory again are used to manage the archives and its directory. Or when using PKUNZIP to view the files, the more files in the archive, the more memory that is required and the more time that is involved when sorting the files in archive properties before displaying or printing the contents.
- Archive updating: When updating a large archive with large file sizes, there will be overhead to copy the files from the previous archive, before adding or updating new files to the archive. For example, if you have a 10 GB archive with 5 files that are each compressed, down to 2 GB, overhead will be required to copy the compressed files from the old archive to the new archive. This is another reason for storing the archive in the IFS, which can help reduce resources rather than storing the archive in a file in a library.
- When compressing large size files, PKZIP will sometimes use as much CPU as the system will allow. With this in mind, processing very large files may perform best as a submitted job. Some IBM i systems have constraints on running interactively, and if interactive jobs run a long time and use high amounts of CPU resources, their system will start slowing down and may issue the message CPI1479 "Interactive activity approaching capacity of installed feature." In this case they should review the details of this message, which usually means that their interactive systems are using more resources than the IBM i was configured to use.

Encryption Performance

Archives using advanced encryption (AES) will be slightly larger (approximately 300 bytes per file in archive) than archives with no encryption. The increase in size will be the same whether you use AES 128, AES 192, or AES 256.

Being the most secure encryption algorithm, AES 256 will also consume the most CPU usage. AES 128 on average could use around 9% more CPU than running with no encryption. AES 256 averages about 3.4% more usage when compared with AES 128 (or around 12.5% versus no encryption).

Extended Attributes Selections

The extended attributes naturally contribute some overhead to the archive but it is minimal, unless you are compressing a database file in the QSYS library file system with the parameter DBSERVICE(*YES). This size then depends on the definitions of the database (fields, headings, etc.), but also is very important in rebuilding a DB2 database where it does not exist.

These extended attributes can be stored in two places, called the local header and central header directories. ***SecureZIP**^{i}* 8.2 and other current PKWARE products now only use the extended attributes from the central directory. To help reduce the archive overhead the parameter EXTRAFLD in ***Smartcrypt**^{i}* has been expanded to select where you want to store the attributes. By using EXTRAFLD(*Central), you reduce the size of each file in the archive by the size of the extended attributes.

B CLP Samples

The following CLP samples can be found in the QCLSRC file of the distributed library. They contain programming source code for your consideration. These samples have not been thoroughly tested under all conditions. PKWARE, Inc, therefore, cannot guarantee or imply reliability or functionality of these programs. The programs contained herein are provided to you "AS IS".

PKSAMP01 – Override for Stdout and Stderr to an OUTQ

This sample demonstrates how to override the PKZIP and PKUNZIP program output, and then redirect the output to an OUTQ. This also provides an example of using mixed file systems, such as having the archive file in the IFS and selecting files from the QSYS library file system.

PKSAMP02 – Compress all files in TESTLIB with PKZIP

This sample demonstrates using PKZIP in a CL passing the archive’s library, file, and member as variables and then monitoring for errors from the PKZIP run.

PKSAMP03 – Capture Last SPLF in Job

This brief CLP sample demonstrates using PKZSPOOL to compress only the last spool file written out by the current job to a PDF.

PKSAMP04 – SBMJOB to Capture all SPLF of Job

This sample performs several tasks that print reports, then submits a job at the end of the job that will compress all spool files to PDF files (including the job log).

PKSAMP05 – Strong Encrypt Calling Password

This sample calls PKSAMP05A to obtain a password for a PKZIP and PKUNZIP process.

PKSAMP05A – Password CL Store for PKSAMP06

This sample is called from PKZIP for IBM i CL that requires a password. The parameter will determine which password to return to the caller PGM.

PKSAMP06 – Creating archives with 1Step2Tape Old

This sample will run two PKZIP commands with the archives being written directly to the tape that is on the inputted tape device. This will destroy the contents on the tape since the first archive will be written to file number 1 on the tape. These tape archive files are created without creating the shadow directory files.

PKSAMP07 – 1Step2Tape with View/Test Tape Input Archive Files

This sample will run three PKZIP commands with the archives being written directly to the tape that is on the inputted tape device. This will destroy the contents on the tape, since the first archive will be written to file number 1 on the tape.

Next a series of PKUNZIP commands (*VIEW, *TEST) will be run to read the newly created archives directly from tape.

PKSAMP08 – Run iPSRA to SAVLIB and Capture Resulting Spool Files

This sample demonstrates the use of iPSRA, using the SAVLIB command. The OUTPUT parameter is then used to print the resulting output. Finally, a PKZIP job is submitted to add the printed spool file (in PDF format) from the SAVLIB command to the current archive.

PKSAMP09 – Lib Encryption to Tape with Multi Steps

This sample specifies a library that will be saved, compressed, encrypted and stored on tape.

PKSAMP10 – 1Step2Tape and iPSRA Multiple Libraries

This sample demonstrates how to save multiple libraries directly to tape utilizing iPSRA and 1Step2Tape features. It simulates the SAVLIB command with multiple libraries by writing each library's saved archive directly to tape using the tape label name as the library name. It also creates a LOG file for each saved library as the member name. This file contains the contents of the save operation.

This sample can be used to simulate using a *NONSYS with a SAVLIB command.

This CL displays library type objects, with output going to a temporary file. The CL then loops through the output file, performs a PKZIP with iPSRA options for each selected library, and creates the archive directly to tape using the tape label for the library name. Encryption could be added to make the save files secure.

PKSAMP11 – Change Ownership of PKWARE Objects

This sample demonstrates how to change the OWNER of all PKWARE objects. Before compiling, you will need to do DSPOBJD to establish the objects file QTEMP/PKOBJS.

C List Files

The list file capabilities provided in the PKZIP and PKUNZIP commands can be a powerful tool for maintaining detailed selection criteria and to exclude files. PKZIP and PKUNZIP commands also allow creating a list of files that are located in a particular archive.

Creating List Files

Both PKZIP and PKUNZIP can create a text format file of file names that meet criteria entered within the FILES and EXCLUDE parameters. In PKZIP, the output files contain the names of all files in the IBM i OS format, depending on if the files are from the QSYS file system or IFS. The PKUNZIP program will produce a list of names in the format of the archive. To create an output list file, place the output file name in the parameter CRTLIST(). The default value is CRTLIST(*NONE).

Note: List File data is stored in EBCDIC.

Depending on the value of the TYPLISTFL parameter, the output file can be put in either the QSYS file system or IFS.

TYPLISTFL(*DB): When the file system is QSYS, the output file will create a physical file (PF-DTA) with a record length of 132. For the file format in CRTLIST, you can use any of the following formats: library/file, library/file(member), file, or file(member). When a member is not entered, the member name will be the same as the file name. You should use the utility that your organization uses to edit data files.

TYPLISTFL(*IFS): When using the IFS, the output file will create a stream file (*STMF object type). Most organization uses EDTF. For the file format in CRTLIST, you can use any of the following formats: file, file.suffix, dir1/file, dir1/dir2/../dirn/file, /dir1., etc. When the path does not start with '/', then the path starts in your current directory (relative path).

When creating a file manually, follow the creation attributes described above.

Using List Files as Input

Both PKZIP and PKUNZIP programs can use list file parameters for both selections of files (INCLFILE(‘file name’)) and/or the excluding of file (EXCLFILE(‘file name’). They can also use an inlist file for the encryption recipient ENTPREC. The file name of parameters depends on the setting of TYPLISTFL (*DB or *IFS) and should follow the guidelines in “Creating List Files,” above.

When using PKZIP, the format of files in the list file should be in the format of the IBM i files that will be processed. See the parameters FILES and EXCLUDE for specifications.

When using PKUNZIP, the format of the files in the list file should be in the format of the archive. See the parameters FILES and EXCLUDE for specifications.

PKUNZIP also has an option to create a list file in expanded mode, which will display the date and time modified along with the file names. This is accomplished by having a ‘>’ character being the in the first position of the CRTLIST parameter. See the two examples below.

Create normal list file:➔ PKUNZIP ARCHIVE('atest/V100/listf') CRTLIST('atest/listfile(demo)')

```
 Edit File: ATEST/LISTFILE(DEMO)
 Record :       1   of       4 by   8           Column :    1    132 by  74
CMD ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+
                ************Beginning of data**************
    TESTLIB/MYFILE/MYMBR
    TESTLIB/MYFILEGE.R/MYMBR
    TESTLIB/MYFILETE.XT/MYMBR
    TESTLIB/MYFILE27.3/MYMBR
                ************End of Data********************
```

Create an expanded list file:➔ PKUNZIP ARCHIVE('atest/V100/listf') CRTLIST('>atest/listfile(demo)')

```
 Edit File: ATEST/LISTFILE(DEMO)
 Record :       1   of       4 by   8           Column :    1    132 by  74
CMD ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+
                ************Beginning of data**************
    DT(05-20-03 16:16)  TESTLIB/MYFILE/MYMBR
    DT(02-14-03 16:44)  TESTLIB/MYFILEGE.R/MYMBR
    DT(02-14-03 16:44)  TESTLIB/MYFILETE.XT/MYMBR
    DT(02-14-03 16:44)  TESTLIB/MYFILE27.3/MYMBR
                ************End of Data********************
```

D Translation Tables

Text files (such as program source code) are usually held within an archive using the ASCII character set for compatibility with other versions of ***PKZIP***. For these to be usable on IBM i OS, they must be converted to the IBM EBCDIC character set. ***Smartcrypt**ⁱ* uses one of two possible internal translation tables, which should be suitable for most customers. These translation table members are used by parameters FTRAN and TRAN in both the PKZIP and PKUNZIP programs. Included (as part of the distribution) are a series of override translation tables. Some users may wish to define their own table.

The override translation tables included are stored as source members in file PKZTABLES ***Smartcrypt**ⁱ* resources tables. By referencing the members in parameters TRAN and FTRAN, ***Smartcrypt**ⁱ* will access the selected member in the PKZTABLES file and parse them to an internal hexadecimal table for use in translation.

The following translation tables are included:

Table Name	*Translation from*	*Translation to*	*Explanation*
ISO9959_1	819 or ISO9959_1	037	ASCII-819 <-> EBCDIC-037 Translation (Same as using *ISO99591)
ASCIIISO	EBCDIC	ASCII - iso	Translate Table
LATIN1	EBCDIC	ASCII	Latin Translate Table
NOOP	NO-OP		Translation Table Straight Hex
UKASCII	EBCDIC	UK	ASCII Translate Table
UKASCIIE	EBCDIC	UK	ASCII Translate Table-Euro
USASCII	EBCDIC	USA	ASCII Translate Table (Same as using *INTERNAL)
USASCIIE	EBCDIC	USA	ASCII Translate Table-Euro

Standard Code Page Support with Tables

Three data translation tables are available to assist with one or more of the latest standard EBCDIC text translation to ASCII. These tables were built to relate directly to IBM code pages numbers.

Code page tables available are:

Table Name	*ASCII Code Page*	*EBCDIC Code Page*	*Explanation*
PKZ819037	819	037	ASCII-819 <-> EBCDIC-037 Translation
PKZ819273	819	273	ASCII-819 <-> EBCDIC-273 Translation German
PKZ819277	819	277	ASCII-819 <-> EBCDIC-277 Translation Den/Nor
PKZ819278	819	278	ASCII-819 <-> EBCDIC-278 Translation Fin/Swe
PKZ819280	819	280	ASCII-819 <-> EBCDIC-0280 Translation Italy
PKZ819284	819	284	ASCII-819 <-> EBCDIC-284 Translation Spanish
PKZ819297	819	297	ASCII-819 <-> EBCDIC-297 Translation French
PKZ819500	819	500	ASCII-819 <-> EBCDIC-500 Translation ISO8859-1
PKZ819871	819	871	ASCII-819 <-> EBCDIC-871 Translation Icelandic
PKZ850037	850	037	ASCII-850 <-> EBCDIC-037 Translation
PKZ850284	850	284	ASCII-850 <-> EBCDIC-284 Translation Spanish

International Code Page Support

Some data-interchange environments require specialized multi-language character translation support. ***Smartcrypt**i* provides tables for character based data translation through translation tables that are also included in the PKZTABLES.

The tables for the following international code pages are provided in the ***Smartcrypt**i* PKZTABLES as members TRTxxyy (where xx = “from” and yy = “to”).

Language	*EBCDIC*	*ASCII*	*EURO/ASCII*	*FROM*	*TO*	*EURO*
German	273	850*	858	EB	AA	AI
Spanish	284	850	858	EJ	AA	AI
Portuguese	282	850	858	EI	AA	AI
Italian	280	850	858	EG	AA	AI
Danish	277	850	858	EE	AA	AI
Norwegian	277	850	858	EE	AA	AI
Swedish	278	850	858	EF	AA	AI
Finnish	278	850	858	EF	AA	AI
French	297	850	858	EM	AA	AI

* IBM-850 = IBM-4946

These members are provided "as is." It is the responsibility of the user to ensure that data translation mapping is in accordance with their business needs.

Translation Table Layout

There are two translation tables in PKZTABLES. The first table is a translation from ASCII to EBCDIC. The second is EBCDIC to ASCII.

In each table there are 256 entries representing hex values from x'00' thru x'FF'.

Each entry is represented as a 4-character field such as 0x00 and 0xFF.

On each line there must be 8 entries with each entry separated by a space. With 8 entries per line, there must be 32 lines of table entries for each table set, representing the 256 translation values.

The tables have embedded comments to help in their documentation.

In the table example below, to translate an ASCII character **A** (hexadecimal x'41' or decimal value of '65'), go to entry 65 in the table (Line 8, entry 2) and find a hexadecimal x'C1' which is the EBCDIC **A**.

See "Example of PKZTABLES (USASCII) Translation Table."

Note: Do not alter any other members found in the PKZTABLES file or ***Smartcrypt****^i^* may not function correctly.

Creating New Translation Table Members

Take the following steps to define your own translation table:

1. Copy one of the distributed members in PKZTABLES to a member name of your choice.
2. Edit the new table using the IBM i OS Source Entry Utility (SEU).
3. Change the values with respect to the layout describe above, making sure not to alter the overall table layout. If the overall layout is altered, ***Smartcrypt****^i^* may not work correctly.
4. Save the member and test your changes.

Example of PKZTABLES (USASCII) Translation Table

```
/* PKZIP/400 Translate Table USASCII to EBCDIC */
/*00-07*/ 0x00 0x01 0x02 0x03 0x37 0x2D 0x2E 0x2F  /*00-07*/
/*08-0f*/ 0x16 0x05 0x25 0x0B 0x0C 0x0D 0x0E 0x9F  /*08-0f*/
/*10-17*/ 0x10 0x11 0x12 0x13 0xB6 0xB5 0x32 0x26  /*10-17*/
/*18-1f*/ 0x18 0x19 0x3F 0x27 0x1C 0x1D 0x1E 0x1F  /*18-1f*/
/*20-27*/ 0x40 0x5A 0x7F 0x7B 0x5B 0x6C 0x50 0x7D  /*20-27*/
/*28-2f*/ 0x4D 0x5D 0x5C 0x4E 0x6B 0x60 0x4B 0x61  /*28-2f*/
/*30-37*/ 0xF0 0xF1 0xF2 0xF3 0xF4 0xF5 0xF6 0xF7  /*30-37*/
/*38-3f*/ 0xF8 0xF9 0x7A 0x5E 0x4C 0x7E 0x6E 0x6F  /*38-3f*/
/*40-47*/ 0x7C 0xC1 0xC2 0xC3 0xC4 0xC5 0xC6 0xC7  /*40-47*/
/*48-4f*/ 0xC8 0xC9 0xD1 0xD2 0xD3 0xD4 0xD5 0xD6  /*48-4f*/
/*50-57*/ 0xD7 0xD8 0xD9 0xE2 0xE3 0xE4 0xE5 0xE6  /*50-57*/
/*58-5f*/ 0xE7 0xE8 0xE9 0xBA 0xE0 0xBB 0xB0 0x6D  /*58-5f*/
/*60-67*/ 0x79 0x81 0x82 0x83 0x84 0x85 0x86 0x87  /*60-67*/
/*68-6f*/ 0x88 0x89 0x91 0x92 0x93 0x94 0x95 0x96  /*68-6f*/
/*70-77*/ 0x97 0x98 0x99 0xA2 0xA3 0xA4 0xA5 0xA6  /*70-77*/
/*78-7f*/ 0xA7 0xA8 0xA9 0xC0 0x6A 0xD0 0xA1 0x07  /*78-7f*/
/*80-87*/ 0x68 0xDC 0x51 0x42 0x43 0x44 0x47 0x48  /*80-87*/
/*88-8f*/ 0x52 0x53 0x54 0x57 0x56 0x58 0x63 0x67  /*88-8f*/
/*90-97*/ 0x71 0x9C 0x9E 0xCB 0xCC 0xCD 0xDB 0xDD  /*90-97*/
/*98-9f*/ 0xDF 0xEC 0xFC 0x4A 0xB1 0xB2 0x3E 0xB4  /*98-9f*/
/*a0-a7*/ 0x45 0x55 0xCE 0xDE 0x49 0x69 0x9A 0x9B  /*a0-a7*/
/*a8-af*/ 0xAB 0x0F 0x5F 0xB8 0xB7 0xAA 0x8A 0x8B  /*a8-af*/
/*b0-b7*/ 0x3C 0x3D 0x62 0x4F 0x64 0x65 0x66 0x20  /*b0-b7*/
/*b8-bf*/ 0x21 0x22 0x70 0x23 0x72 0x73 0x74 0xBE  /*b8-bf*/
/*c0-c7*/ 0x76 0x77 0x78 0x80 0x24 0x15 0x8C 0x8D  /*c0-c7*/
/*c8-cf*/ 0x8E 0x41 0x06 0x17 0x28 0x29 0x9D 0x2A  /*c8-cf*/
/*d0-d7*/ 0x2B 0x2C 0x09 0x0A 0xAC 0xAD 0xAE 0xAF  /*d0-d7*/
/*d8-df*/ 0x1B 0x30 0x31 0xFA 0x1A 0x33 0x34 0x35  /*d8-df*/
/*e0-e7*/ 0x36 0x59 0x08 0x38 0xBC 0x39 0xA0 0xBF  /*e0-e7*/
/*e8-ef*/ 0xCA 0x3A 0xFE 0x3B 0x04 0xCF 0xDA 0x14  /*e8-ef*/
/*f0-f7*/ 0xE1 0x8F 0x46 0x75 0xFD 0xEB 0xEE 0xED  /*f0-f7*/
/*f8-ff*/ 0x90 0xEF 0xB3 0xFB 0xB9 0xEA 0xBD 0xFF  /*f8-ff*/

/* PKZIP/400 Translate Table EBCDIC to USASCII */
/*00-07*/ 0x00 0x01 0x02 0x03 0xEC 0x09 0xCA 0x7F  /*00-07*/
/*08-0f*/ 0xE2 0xD2 0xD3 0x0B 0x0C 0x0D 0x0E 0xA9  /*08-0f*/
/*10-17*/ 0x10 0x11 0x12 0x13 0xEF 0xC5 0x08 0xCB  /*10-17*/
/*18-1f*/ 0x18 0x19 0xDC 0xD8 0x1C 0x1D 0x1E 0x1F  /*18-1f*/
/*20-27*/ 0xB7 0xB8 0xB9 0xBB 0xC4 0x0A 0x17 0x1B  /*20-27*/
/*28-2f*/ 0xCC 0xCD 0xCF 0xD0 0xD1 0x05 0x06 0x07  /*28-2f*/
/*30-37*/ 0xD9 0xDA 0x16 0xDD 0xDE 0xDF 0xE0 0x04  /*30-37*/
/*38-3f*/ 0xE3 0xE5 0xE9 0xEB 0xB0 0xB1 0x9E 0x1A  /*38-3f*/
/*40-47*/ 0x20 0xC9 0x83 0x84 0x85 0xA0 0xF2 0x86  /*40-47*/
/*48-4f*/ 0x87 0xA4 0x9B 0x2E 0x3C 0x28 0x2B 0xB3  /*48-4f*/
/*50-57*/ 0x26 0x82 0x88 0x89 0x8A 0xA1 0x8C 0x8B  /*50-57*/
/*58-5f*/ 0x8D 0xE1 0x21 0x24 0x2A 0x29 0x3B 0xAA  /*58-5f*/
/*60-67*/ 0x2D 0x2F 0xB2 0x8E 0xB4 0xB5 0xB6 0x8F  /*60-67*/
/*68-6f*/ 0x80 0xA5 0x7C 0x2C 0x25 0x5F 0x3E 0x3F  /*68-6f*/
/*70-77*/ 0xBA 0x90 0xBC 0xBD 0xBE 0xF3 0xC0 0xC1  /*70-77*/
/*78-7f*/ 0xC2 0x60 0x3A 0x23 0x40 0x27 0x3D 0x22  /*78-7f*/
/*80-87*/ 0xC3 0x61 0x62 0x63 0x64 0x65 0x66 0x67  /*80-87*/
/*88-8f*/ 0x68 0x69 0xAE 0xAF 0xC6 0xC7 0xC8 0xF1  /*88-8f*/
/*90-97*/ 0xF8 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F 0x70  /*90-97*/
/*98-9f*/ 0x71 0x72 0xA6 0xA7 0x91 0xCE 0x92 0x0F  /*98-9f*/
/*a0-a7*/ 0xE6 0x7E 0x73 0x74 0x75 0x76 0x77 0x78  /*a0-a7*/
/*a8-af*/ 0x79 0x7A 0xAD 0xA8 0xD4 0xD5 0xD6 0xD7  /*a8-af*/
/*b0-b7*/ 0x5E 0x9C 0x9D 0xFA 0x9F 0x15 0x14 0xAC  /*b0-b7*/
/*b8-bf*/ 0xAB 0xFC 0x5B 0x5D 0xE4 0xFE 0xBF 0xE7  /*b8-bf*/
/*c0-c7*/ 0x7B 0x41 0x42 0x43 0x44 0x45 0x46 0x47  /*c0-c7*/
/*c8-cf*/ 0x48 0x49 0xE8 0x93 0x94 0x95 0xA2 0xED  /*c8-cf*/
/*d0-d7*/ 0x7D 0x4A 0x4B 0x4C 0x4D 0x4E 0x4F 0x50  /*d0-d7*/
/*d8-df*/ 0x51 0x52 0xEE 0x96 0x81 0x97 0xA3 0x98  /*d8-df*/
/*e0-e7*/ 0x5C 0xF0 0x53 0x54 0x55 0x56 0x57 0x58  /*e0-e7*/
/*e8-ef*/ 0x59 0x5A 0xFD 0xF5 0x99 0xF7 0xF6 0xF9  /*e8-ef*/
/*f0-f7*/ 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37  /*f0-f7*/
/*f8-ff*/ 0x38 0x39 0xDB 0xFB 0x9A 0xF4 0xEA 0xFF  /*f8-ff*/
/* PKZIP/400 Translate Tables end */
```

E Spool File Considerations

This appendix contains information on how ***Smartcrypt**ⁱ* handles spool files in different scenarios that might be helpful to consider in planning for compressing spool files.

Spool File Selections

Be aware that if you set all of the spool file selection parameters to *ALL, you will select all spool files on your IBM i system. This is why the default for the user ID is SFUSER(*CURRENT) to at least limit it to the current user in case a selection is not filled in correctly.

If a spool file is deleted after the selection but before the actual compression takes place, the PKZIP job will fail.

SPLF Attributes

When a spool file is selected and the parameter EXTRAFL is coded *YES (the default), then the extended attributes listed below are stored in archive and can be viewed with PKUNZIP TYPE(*VIEW) VIEWOPT(*ALL). Also when the spool files are stored in the archive, the date and time for the file is the spool files creation date and time and can be viewed with PKUNZIP.

Extended Attributes:

- Description: The spool file description is built as follows:

  "Job-Name/User-Name/#Job-Number/Spool-File-Name/Fspool-File-Number.Suffix" For Example: "MYJOB/BILLS#152681/INVOICE/F0021.SPLF"
- Spool file type: *SCS: SNA Character Stream, *IPDS: An intelligent printer data stream, *AFPDS: Advanced Function Print Data Stream, *USERASCII: An ASCII data stream user defined, *LINE: Line data that is very printer specific, and *AFPDSLINE: Mixed data (line data and AFPDS data).
- Target file created: Describes the target type file created during compression. SPLF: Spool Files, TXT: ASCII Text Conversion, and PDF: Portable Document Format.
- Number of pages contained in the spool file.

An example of the attributes view seen with –VIEWOPT(*ALL) for a spool file converted to a PDF might appear as follows:

```
Filename: CRTCM60.PDF
Detected File type:     Binary
Created by:             PKZIP for IBM i   16.0   PKZIP 2.x compatible
Minimum to Extract:     PKZIP 2.0  Or Greater
Compression method:     Deflated   [Fast]
Date and Time           2009 Oct 17 07:22:00
Compressed size:        2316 bytes
Uncompressed size:      8146 bytes
32-bit CRC value (hex): 40950039
Extended attributes:    yes, [Length = 112]
Spool File Type:*SCS, Target File:PDF, Nbr Of pages(3).
SPLF Desc:USER1/USER1/#007892/CRTCM/F0060.PDF.
File Comment:"none"
```

The preceding view comes from the following spool file:

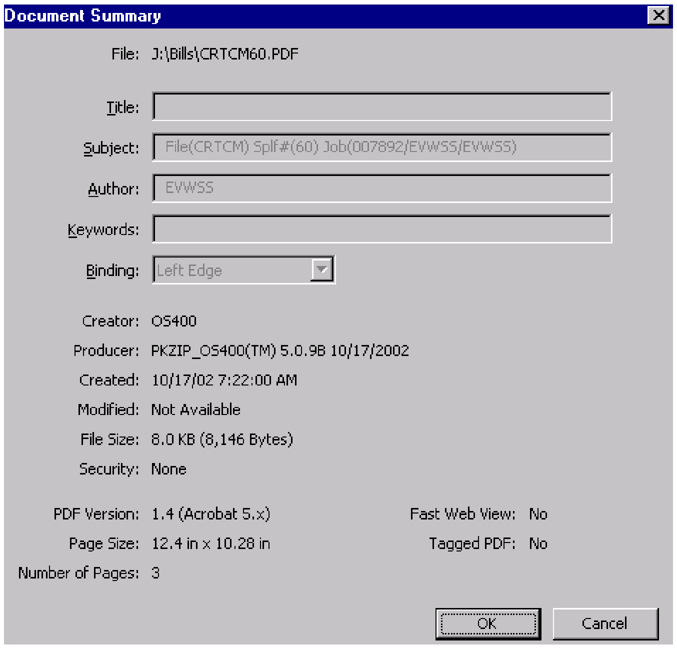
```
5722SS1  V5R1M0   Work With Output Queue    QPRINT2  in  QGPL  11/19/02 14:08:53 Page    1
File   User  User Data Status Pages  Cpy Form Tp  Pty File Number  Job    Number Date
Time
CRTCM  USER1            RDY      3    1   *STD     5         60    USER1  007892 10/17/09
07:22:00
```

PDF Creation Attributes

When creating a PDF, the attributes are also stored in the PDF document to help trace back what spool file they originated from.

- The date and time of the spool file creation will be the PDF date and time of creation.
- The author will be the user ID that created the spool file.
- The subject will be made up of the spool file name, number, and the job (number/user ID/job name).

An example of a PDF summary is:



File: J:\Bills\CRTCM60.PDF

Title:

Subject: File(CRTCM) Splf#(60) Job(007892/EVWSS/EVWSS)

Author: EVWSS

Keywords:

Binding: Left Edge

Creator: OS400

Producer: PKZIP_OS400(TM) 5.0.9B 10/17/2002

Created: 10/17/02 7:22:00 AM

Modified: Not Available

File Size: 8.0 KB (8,146 Bytes)

Security: None

PDF Version: 1.4 (Acrobat 5.x)   Fast Web View: No

Page Size: 12.4 in x 10.28 in   Tagged PDF: No

Number of Pages: 3

F Contact Information

PKWARE, Inc.

Web Site: www.pkware.com

For Licensing, please contact the Sales Division at 937-847-2374 or email PKSALES@PKWARE.COM.

For Technical Support assistance, please contact the Product Services Division at 937-847-2687 or visit the Support Web site.

PROBLEM REPORTING

Providing appropriate documentation on the initial call for a problem expedites the analysis and resolution process. The following sections describe the type of information that should be supplied for each category of problem.

PROBLEM REPORTING (General)

When reporting a problem regarding ***Smartcrypt****i*, please be prepared to provide the following information:

- The displayed output from ➔**CALL ziplib / WHATOSV** or the details that WHATOSV provides
- Release level of the operating system
- Release level of PKZIP for IBM i being run
- A description of the process being run and any differentiating circumstances from job(s) that do run
- A display of the command problem with parameters
- A copy of the output and JobLog from the failing execution
- If run from a CL and practical, please include source listing of the CL
- If PKUNZIP is failing, provide the Output from the following:

➔ **PKUNZIP TYPE(*VIEW) VIEWOPT(*ALL)**

- If requested by Technical Support, the display with various tracing options turned on
- If practical, please include the archive/input file involved in the failing execution

PROBLEM REPORTING (Licensing)

When reporting a problem regarding licensing, please be prepared to provide the following information:

- The displayed output from ➔**CALL ziplib / WHATOSV**
- A copy of the INSTPKLIC command and its parameters
- A copy of the output from the INSTPKLIC job

If the problem occurs in a ***Smartcrypt**^{i}* job, follow the steps outlined above.

G Options for Running Self-Extracting Archives

Self-extracting (SFX) archives are executable files that do not require an external program to extract their contents. To extract files from a self-extracting archive, you just run the archive.

Self-extracting archives can be created for various UNIX platforms, for Linux, and for Windows. Windows self-extractors can be created either for command line execution or to run in the Windows graphical interface.

By default, running a self-extractor extracts all its files. But most self-extracting archives can be executed with a number of options. Some options set filter conditions that constrain the set of files to be extracted. Other options affect such things as whether to extract saved paths with the files, whether to overwrite similarly named files at the destination, and so on.

This appendix lists options that can be used when executing either command line or graphical (Windows) self-extractors. In the case of the command line self-extractors, slightly different sets of options are available for self-extractors created with different versions of the SFX utility, namely, versions 2.5, 6.1, 10 and 12. The differences are flagged in the table that describes the options, in this appendix.

To find out the version of a command line self-extractor, run it with the `-h` option:

```
myselfextractor.exe -h
```

This displays information on the screen; it does not extract any files.

Command Line Self-Extractors

This section describes options available with SFX 2.5, 6.1, 10 and 12 command line self-extractors. These self-extracting archives are created to be executed on the command line. Specify the options in the command line used to run the self-extractor.

Usage

The syntax for executing a self-extracting archive is:

```
<sfx.exe> [options] [files...]
```

where

- `<sfx.exe>` (without the brackets) is the name of the self-extracting archive to be executed
- `options` is a list of options to apply (separated by spaces if more than one). Prefix each option name with a hyphen "–". For example: `sfx.exe -c`
- `files` is a list of files or file name specifications (separated by spaces if more than one) denoting the files to be extracted from the archive. If no files are specified, all files are extracted.

By default, if no options are used all files are extracted when you run a self-extracting archive.

Abbreviate Option Names

On the command line, option names can be abbreviated (truncated) as long as they unambiguously pick out an option. For example, instead of

mysfx.exe –lowercase

you can enter

mysfx.exe –low

but not

mysfx.exe –l

or

mysfx.exe –lo

because the latter two could also refer to other options (`license`, `locale`).

Options for Command Line Self-Extractors

The table below lists command line extraction options for SFX 2.5, 6.1, 10 and 12 self-extractors. With a few exceptions, the different SFX versions offer identical options. The exceptions are flagged in the table.

Some options have suboptions. For example, the `after` option has a suboption in which to specify a date. Prefix a suboption value with an equal sign (=) and enter it immediately after the option (no spaces): `-after=12312006`.

Option	***Description***
after	Extracts files that are newer than or equal to a specified date Suboptions: A date [format: mmddyy or mmddyyyy] Example: `mysfx.exe -after=12312006`
before	Extracts files that are older than a specified date Suboptions: A date [format: mmddyy or mmddyyyy] Example: `mysfx.exe -before=12312006`

Option	***Description***
console	Displays the contents of specified archived files on your screen Example: `mysfx.exe -console readme.txt`
directories	Recreates directory path, including any sub-directories Example: `mysfx.exe -dir`
exclude	Excludes specified files from being extracted Example: `mysfx.exe -exclude="*.txt"`
extract	Extracts only files that satisfy the condition in the suboption. Suboptions: *all* [extract everything in archive] *freshen* [extract if newer than an existing destination copy] *update* [extract if newer or not in destination directory] Example: `mysfx.exe -extract=freshen`
fipsmode SFX 12 or later	Enable FIPS mode Example: `mysfx.exe -fipsmode`
help	Displays help screen listing available options Example: `mysfx.exe -help`
id	(UNIX only) Preserve original file uid/gid ownership. Must be root/file owner Suboptions: *Userid* [Restore user ownership] *Groupid* [Restore group ownership] *None* [Do not restore ownership] *All* [Same as specifying *userid* and *groupid*] Example: `mysfx.exe -id=userid`
include	Includes specified files for extraction. (This option is ordinarily not necessary. All specified files are included anyway.) Example: `mysfx.exe -include="*.txt"`
keypassphrase UNIX only SFX 12 or later	Specifies a passphrase for a private key in the certificate store Example: `mysfx.exe –` keypassphrase=secret
larger SFX 6.1 and later	Extracts files that are the specified size (in bytes) and larger Suboptions: <A numerical value (in bytes) that indicates a minimum desired file size> Example: `mysfx.exe -larger=400`
license	Displays license information Example: `mysfx.exe -license`

Option	***Description***
locale	Reads and/or adjusts the locale variable for date and time format input Suboptions: *Enable* [Use current locale] *Disable* [Use US locale] Example: `mysfx.exe -locale=disable -after=12312006`
lowercase	Changes file names to lower case on extraction Example: `mysfx.exe -lowercase`
mask	(Windows only) Removes specified file attributes upon extraction Suboptions: *archive* [mask archive attribute from file(s)/folder(s)] *hidden* [mask hidden attribute from file(s)/folder(s)] *system* [mask system attribute from file(s)/folder(s)] *readonly* [mask read-only attribute from file(s)/folder(s)] *none* [do not mask attributes from file(s)/folder(s)] *all* [mask all attributes from file(s)/folder(s)] Example: `mysfx.exe -mask=archive,readonly`
mask	(UNIX only) Removes specified file permissions upon extraction. Suboptions: <Octal mode value of permissions to be removed> Example: `mysfx.exe –mask=077`
more	Displays output one screen at a time Example: `mysfx.exe -more`
newer SFX 6.1 and later	Extracts only those files that are newer than a specified (calendar) day in the past Suboptions: <Number of calendar days ago to set cutoff date> Example: `mysfx.exe -newer=2`
noextended SFX 2.5 and 6.1 only	Suppresses the extraction of extended permission and timestamp attributes Example: `mysfx.exe -noextended`
older SFX 6.1 and later	Extracts only those files that are older than a specified (calendar) day in the past Suboptions: <Number of calendar days ago to set cutoff date> Example: `mysfx.exe -older=2`

Option	*Description*
overwrite	Overwrites existing files Suboptions: *prompt* [prompt before overwriting] *all* [always overwrite] *never* [never overwrite] Example: `mysfx.exe -overwrite=all`
password	Specifies a decryption password Example: `mysfx.exe -password=grendel`
passphrase SFX v10 and later	Specifies a decryption passphrase (or password) Example: `mysfx.exe -passphrase=grendel`
permission UNIX only	Sets additional permissions on the files being extracted. Suboption: <Octal mode value of permissions to add> Example: `mysfx.exe –permission=111`
print Windows only	Prints the specified archived file Suboptions: <print device name> [for example print=lpt1] Example: `mysfx.exe -print=lpt2 readme.txt`
silent	Suppresses warning messages when extracting Example: `mysfx.exe -silent`
smaller SFX 6.1 and later	Extracts files that are the specified size (in bytes) and smaller Suboptions: <A numerical value (in bytes) that indicates a maximum desire file size> Example: `mysfx.exe -smaller=400`
sort	Sorts files when extracting Suboptions: *crc* [sort by crc value] *date* [sort by date of the file] *extension* [sort by file extension] *name* [sort by file name] *natural* [sort in the order that the file was archived] *ratio* [sort by compression ratio] *size* [sort by file size] *none* [do not sort] Example: `mysfx.exe -sort=size`

Option	***Description***
test	Tests the integrity of archived files Suboptions: *all* [test everything in archive] *freshen* [test if newer than destination copy] *update* [test if newer or not in destination directory] Example: `mysfx.exe -test=all`
times	Preserves specified file date/time stamp Suboptions: *access* [preserve accessed date/time stamp on extraction] *modify* [preserve modified date/time stamp on extraction] *create* [preserve created date/time stamp on extraction] *all* [preserve all date/time stamps on extraction] *none* [do not preserve date/time stamps on extraction] Example: `mysfx.exe -time=access,modify`
translate	Translate the end of line sequence for give operating system. EBCDIC options limited to files using Zip Descriptor Word (ZDW) suboptions: *DOS* [convert to DOS style line endings] *MAC* [convert to MAC style line endings] *unix* [convert to unix style line endings] *EBCDIC,NL* [convert to EBCDIC NL - 0x15] *EBCDIC,LF* [convert to EBCDIC LF - 0x25] *EBCDIC,CRLF* [convert to EBCDIC CRLF - 0x0D25] *EBCDIC,LFCR* [convert to EBCDIC LFCR - 0x250D] *EBCDIC,CRNL* [convert to EBCDIC CRNL - 0x0D15] Example: `mysfx.exe -translate=unix`
version	Displays SFX version and return appropriate value to the shell Suboptions: *major* [return major version number] *minor* [return minor version number] *step* [return step or patch version number] Example: `mysfx.exe -version=step`
warning	Prompts whether to continue after a warning message Example: `mysfx.exe -warning`

Windows Graphical Self-Extractors

Some self-extracting archives created for the Windows graphical interface offer options that control how files are extracted. The options are presented in a dialog when the self-extracting archive is run. Check boxes control whether the option is turned on. Choose **OK** to close the dialog and run the self-extractor with the options selected.

The table below describes the options that may be offered. Not all options are offered with every Windows self-extractor.

Option	***Description***
Display messages	Displays any warning messages (but not error messages) in a dialog and suspends extraction until the user clicks a button to acknowledge.
Create error log	Creates an ASCII text file listing any errors encountered during extraction. The file is named `pkerrlog.txt` and is saved in the destination directory.
Create subfolders	Recreates subfolders on any saved paths, starting from the destination folder, when files are extracted.
Create program group	Creates a program group of shortcuts and adds it to the Windows Start menu. Choose the **Group…** button to define a program group for the SFX to create.
Run after extraction	Specifies a script or application to run or a file to display after an SFX archive is extracted.

Glossary

This glossary provides definitions for items that may have been referenced in the PKZIP® documentation. It is not meant to be exhaustive. One Web site that provides excellent source documentation for computing terms is the IBM Terminology site:

http://www-01.ibm.com/software/globalization/terminology/index.jsp

Absolute Path Name

A string of characters used to refer to an object, starting at the highest level (or root) of the directory hierarchy. The absolute path name must begin with a slash (/), which indicates that the path begins at the root. This is in contrast to a Relative Path Name. See also Path Name.

Advanced Encryption Standard (AES)

The Advanced Encryption Standard is the official US Government encryption standard for customer data.

American Standard Code for Information Interchange (ASCII)

The ASCII code (American Standard Code for Information Interchange) was developed by the American National Standards Institute for information exchange among data processing systems, data communications systems, and associated equipment and is the standard character set used on MS-DOS and UNIX-based operating systems. In a ZIP archive, ASCII is used as the normal character set for compressed text files. The ASCII character set consists of 7-bit control characters and symbolic characters, plus a single parity bit. Since ASCII is used by most microcomputers and printers, text-only files can be transferred easily between different kinds of computers and operating systems. While ASCII code does include characters to indicate backspace, carriage return, etc., it does not include accents and special letters that are not used in English. To accommodate those special characters, extended ASCII has additional characters (128-255). Only the first 128 characters in the ASCII character set are standard on all systems. Others may be different for a given language set. It may be necessary to create a different translation tables (see Translation Table) to create standard translation between ASCII and other character sets.

American National Standards Institute (ANSI)

An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

Application Programming Interface (API)

An interface between the operating system (or systems-related program) that allows an application program written in a high-level language to use specific data or services of the operating system or the program. The API also allows you to develop an application program written in a high level language to access PKZIP data and/or functions of the PKZIP system.

Archive

(1) The act of transferring files from the computer into a long-term storage medium. Archived files are often compressed to save space.

(2) An individual file or group of files which must be extracted and decompressed in order to be used.

(3) A file stored on a computer network, which can be retrieved by a file transfer program (FTP) or other means.

(4) The PKZIP file that holds the compressed/zipped data file.

ASCII

See American Standard Code for Information Interchange.

Binary File

A file that contains codes that are not part of the ASCII character set. Binary files can utilize all 256 possible values for each byte in the file.

Code Page

A specification of code points for each graphic character set or for a collection of graphic character sets. Within a given code page, a code point can have only one specific meaning. A code page is also sometimes known as a code set.

Command Line

The blank line on a display console where commands, option numbers, or selections can be entered.

Control Language (CL) Program

A program that is created from source statements consisting entirely of control language commands.

Cryptography

(1) A method of protecting data. Cryptographic services include data encryption and message authentication.

(2) In cryptographic software, the transformation of data to conceal its meaning; secret code.

(3) The transformation of data to conceal its information content, to prevent its undetected modification, or to prevent its unauthorized use.

Current Library

The library that is specified to be the first user library searched for objects requested by a user. The name for the current library can be specified on the sign-on display or in a user profile. When you specify an object name (such as the name of a file or program) on a command, but do not specify a library name, the system searches the libraries in the system part of the library list, then searches the current library before searching the user part of the library list. The current library is also the library that the system uses when you create a new object, if you do not specify a library name.

Cyclic Redundancy Check (CRC)

A Cyclic Redundancy Check is a number derived from a block of data, and stored or transmitted with the data in order to detect any errors in transmission. This can also be used to check the contents of a ZIP archive. It's similar in nature to a checksum. A CRC may be calculated by adding words or bytes of the data. Once the data arrives at the receiving computer, a calculation and comparison is made to the value originally transmitted. If the calculated values are different, a transmission error is indicated. The CRC information is called redundant because it adds no significant information to the transmission or archive itself. It’s only used to check that the contents of a ZIP archive are correct. When a file is compressed, the CRC is calculated and a value is calculated based upon the contents and using a standard algorithm. The resulting value (32 bits in length) is the CRC that is stored with that compressed file. When the file is decompressed, the CRC is recalculated (again, based upon the extracted contents), and compared to the original CRC. Error results will be generated showing any file corruption that may have occurred.

Data Compression

The reduction in size (or space taken) of data volume on the media when performing a save or store operations.

Data Integrity

(1) The condition that exists as long as accidental or intentional destruction, alteration, or loss of data does not occur.

(2) Within the scope of a unit of work, either all changes to the database management systems are completed or none of them are. The set of change operations are considered an integral set.

Double-byte Character Set (DBCS)

A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, displaying, and printing of DBCS characters requires hardware and programs that support DBCS. Four double-byte character sets are supported by the system: Japanese, Korean, Simplified Chinese, and Traditional Chinese. See also the Single-Byte Character Set (SBCS).

Encryption

The transformation of data into an unintelligible form so that the original data either cannot be obtained or can be obtained only by decryption.

Extended Attribute

Information attached to an object that provides a detailed description about the object to an application system or user.

Extended Binary Coded Decimal Interchange Code (EBCDIC)

The Extended Binary Coded Decimal Interchange Code a coded character set of 256 8 bit characters. EBCDIC is similar in nature to ASCII code, which is used on many other computers. When ZIP programs compress a text file, they translate data from EBCDIC to ASCII characters within a ZIP archive using a translation table.

File Transfer Protocol (FTP)

In TCP/IP, an application protocol used for transferring files to and from host computers. FTP requires a user ID and possibly a passphrase to allow access to files on a remote host system. FTP assumes that the transmission control protocol (TCP) is the underlying protocol.

GNU Privacy Guard (GPG)

A program similar to PGP that follows the OpenPGP standard and is therefore compatible with PGP. It was originally written for UNIX and UNIX-like systems, but has been ported to other systems, including Windows and DOS.

GZIP

GZIP (also known as GNU zip) is a compression utility designed to utilize a different standard for handling compressed file data in an archive.

Integrated File System (IFS)

A function of the operating system that provides storage support similar to personal computer operating systems (such as DOS and OS/2) and UNIX systems.

Interactive Job

A job started for a person who signs on to a work station and communicates (or “converses”) with another computing entity such as a mainframe or IBM i system. This is in contrast to a Batch Job.

IBM i Object

An object that exists in a library on the IBM i system and is represented by an object on the PC. For example, a user profile is an IBM i object represented on the PC by the user profile object.

Lempel-Ziv (LZ)

A technique for compressing data. This technique replaces some character strings, which occur repeatedly within the data, with codes. The encoded character strings are then kept in a common dictionary, which is created as the data is being sent.

Library List

A list that indicates which libraries are to be searched and the order in which they are to be searched. The system-recognized identifier is *LIBL.

Logical Partition (LPAR)

A subset of a single IBM i system that contains resources (such as processors, memory, and input/output devices). A logical partition operates as an independent system. If hardware requirements are sufficient, multiple logical partitions can exist within a system.

New ZIP Archive

A new ZIP archive is the archive created by a compression program when either an old ZIP archive is updated or when files are compressed and no ZIP archive currently exists. It may be thought of as the “receiving” archive. Also see Old ZIP archive.

Null Value

A parameter position within a record for which no value is specified.

n-way Processor Architecture

A processor architecture that provides expandability for future system growth by allowing for additional processors. To the user, the additional processors are transparent because they separately manage the work load by sharing the work evenly among the n-way processors.

Old ZIP Archive

An old ZIP archive is an existing archive which is opened by a compression program to be updated or for its contents to be extracted. It may be thought of as the “sending” archive. Also see New ZIP archive.

OpenPGP / RFC 4880

A standard describing files that are compatible with modern versions of Pretty Good Privacy and other, similar programs. This standard is defined in RFC 4880, which superseded the earlier standard, RFC 2440.

Output Queue

An AS/400 object that contains entries for spooled output files to be written to an output device.

Packed Decimal Format

A decimal value in which each byte within a field represents two numeric digits except the far right byte, which contains one digit in bits 0 through 3 and the sign in bits 4 through 7. For all other bytes, bits 0 through 3 represent one digit; bits 4 through 7 represent one digit. For example, the decimal value +123 is represented as 0001 0010 0011 1111 (or 123F in hexadecimal).

Passphrase

A sentence, phrase, or random string of characters that may include spaces and other non-alphabetical characters that serves as a password. The term *password* implies a single, recognized name or word from the dictionary. The term *passphrase* is meant as encouragement to use longer, more varied strings as passwords. Longer passwords—or passphrases—consisting of random strings that contain spaces and other such characters are much more secure than typical passwords of six to eight characters that use words from the dictionary.

Path Name

(1) A string of characters used to refer to an object. The string can consist of one or more elements, each separated by a slash (/), and may begin with a slash. Each element is typically a directory or equivalent, except for the last element, which can be a directory or another object (such as a file).

(2) A sequence of directory names followed by a file name, each separated by a slash.

(3) In a hierarchical file system (HFS), the name used to refer to a file or directory. The path name must start with a slash (/) and consist of elements separated by a slash. The first element must be the name of a registered file system. All remaining elements must be the name of a directory, except the last element, which can be the name of a directory or file. See also Absolute Path Name and Relative Path Name.

(4) The name of an object in the Integrated File System. Protected objects have one or more path names.

Physical File

Describes how data is to be presented to (or received from) a program and how data is stored in the database. A physical file contains a single record format and at least one member.

Pretty Good Privacy (PGP)

The name of a program originally from the 1980s by Phil Zimmerman. It has been updated with more modern encryption algorithms, and made to comply with the OpenPGP standard, RFC 4880.

Production Library

A library which contains objects needed for normal processing. This contrasts with a Test Library.

QSYS

The library shipped with the IBM i system that contains objects, such as authorization lists and device descriptions created by a user, and the system commands and other system objects required to run the system. The system identifier is QSYS.

Qualified Name

The full name of the library that contains the object and the name of the object.

Relative Path Name

A string of characters that is used to refer to an object, starting at some point in the directory hierarchy other than the root. A relative path name does not begin with a slash (/). The starting point is frequently a user's current directory. This is in contrast to an Absolute Path Name. See also Path Name.

Return Code

A value generated by operating system software to a program to indicate the results of an operation by that program. The value may also be generated by the program and passed back to the operator.

Single-Byte Character Set (SBCS)

A coded character set in which each character is represented by a one-byte code point. A one-byte code point allows representation of up to 256 characters. Languages that are based on an alphabet, such as the Latin alphabet (as contrasted with languages that are based on ideographic characters) are usually represented by a single-byte coded character set. For example, the Spanish language can be represented by a single-byte coded character set. See also the Double-Byte Character Set (DBCS).

Source File

A file of programming code that has not yet been compiled into machine language. A source file can be created by the specification of FILETYPE(*SRC) on the create command. A source file can contain source statements for such items as high-level language programs and data description specifications. Source files maintained on a PC typically use a

.TXT as the extension. On a mainframe, source files are typically found in a partitioned data set or are maintained within a library management tool.

Spool File

Files that exist in an "output queue" which contain reports to printed on the AS/400 system. These files along with attributes can then be directed and transformed to a printer attached to your system.

Stream File

A data file that contains continuous streams of bits such as PC files, documents, and other data stored in IBM i folders. Stream files are well suited for storing strings of data such as the text of a document, images, audio, and video. The content and format of stream files are managed by the application rather than by the system.

System Library

The library shipped with the operating system that contains objects such as authorization lists and device descriptions created by a user. Also included are system commands and other system objects required to run the system. The system identifier is QSYS.

Translation Table

Translation tables are used by the PKZIP and PKUNZIP programs for translating characters in compressed text files between the ASCII character sets used within a ZIP archive and the EBCDIC character set used on IBM-based systems. These tables may be created and modified by the user as documented in the User's Guide.

Trigger

A set of predefined actions that run automatically whenever a specified action or change occurs, for example, a change to a specified table or file. Triggers are often used to automate environments, such as running a backup when a certain number of transactions are processed.

Truncate

To cut off or delete the data that will not fit within a specified line width or display. This may also be attributed to data that does not fit within the specified length of a field definition.

User Interface

The actions or items that allow a user to interact with (and/or perform operations on) a computer.

ZIP64

ZIP64 is reference to the archive format that supports more than 65,534 files per archive, uncompressed files greater than 4 Gig and archives greater than 4 Gig.

ZIP Archive

A ZIP archive is used to refer to a single file that contains a number of files compressed into a much smaller physical space by the ZIP software.

Index