PKZIP® /Smartcrypt® for IBM i®

System Administrator's Guide

SZISA-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.

Like ***PKZIP for IBM i***, ***Smartcrypt for IBM i*** provides data compression and data protection on the AS/400, iSeries, i5, and IBM i. But ***Smartcrypt for IBM i*** protects data with digital signatures as well as trusted OpenSSL encryption, either passphrase- or certificate-based, with key lengths of up to 256 bits. Again 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.

Notices

Licensing requirements have changed for this release. See “Licensing Requirements” in Chapter 2 for details.

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* (bold-italicized) 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, iSeries, 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 ***Smartcrypt**i* products include:

- **System Messages:** This manual documents messages issued by the iSeries 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 and 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-P000-00):** 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.

Related Information on the Internet

PKWARE, Inc.

www.pkware.com

FTP site

- ***Smartcrypt Partner for IBM i***
  ftp://bigiron.pkware.com/pub/products/partnerlink/i5OS

National Institute of Standards and Technology

Computer Security Resource Center - http://csrc.ncsl.nist.gov

Information on the AES development -
http://csrc.nist.gov/encryption/aes

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

User Help and Contact Information

For licensing, please contact Sales at 937-847-2374 (888-4PKWARE / 888-475-9273) or email pksales@pkware.com.

For technical assistance, contact Technical Support at 937-847-2687 or visit the support web site: http://www.pkware.com/support/system-i

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

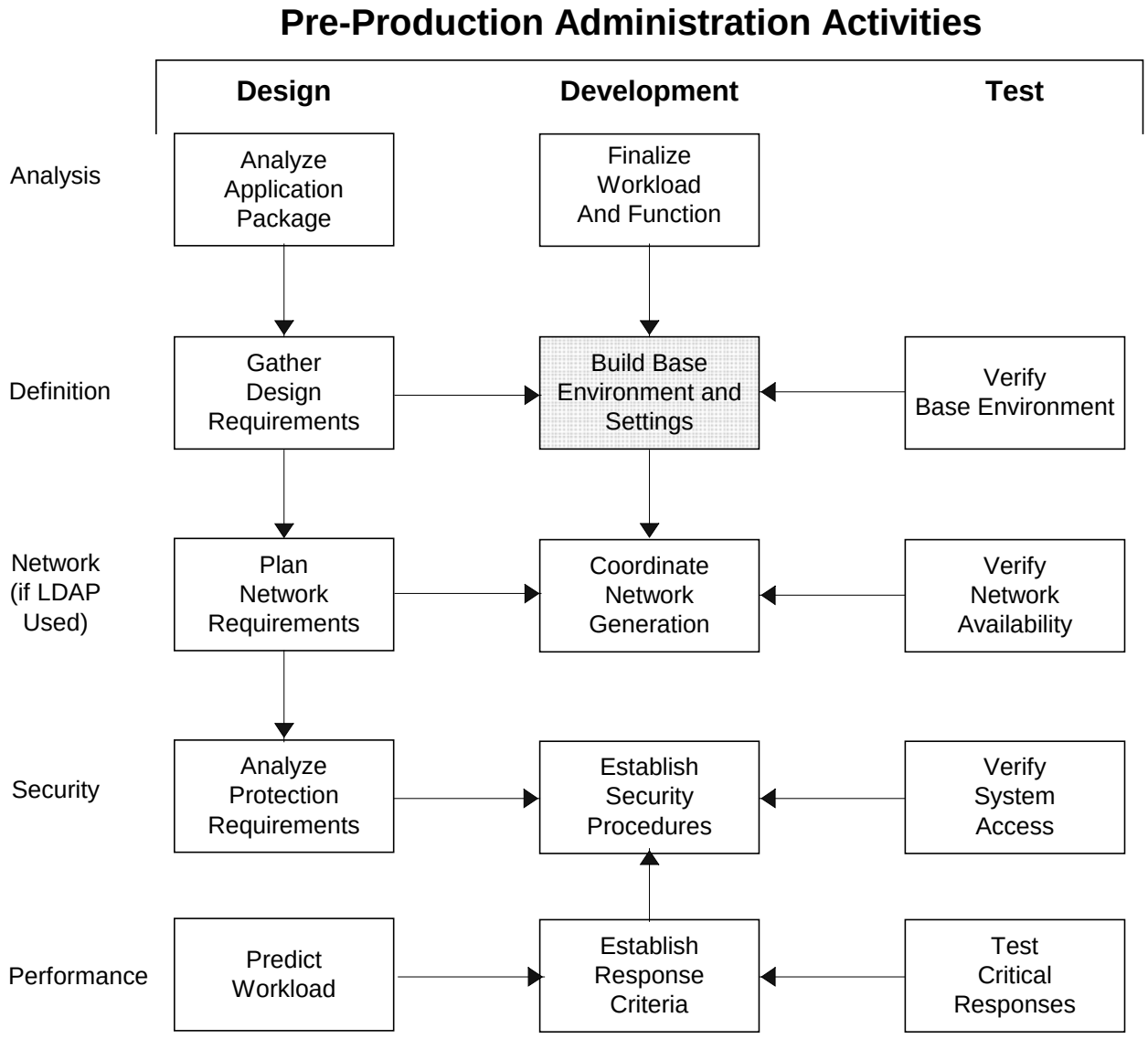
1 System Planning and Administration

Requires Smartcrypt

This chapter deals with installing and administrating ***Smartcrypt for IBM i***. System administrators must plan in advance the design, development, and testing tasks required to integrate ***Smartcrypt for IBM i*** as a secure solution into a production environment. The topics discussed in this chapter relate only to ***Smartcrypt for IBM i***. If you are using ***PKZIP for IBM i***, you may skip this chapter.

The following sections chart the pre-production planning activities for administration and discuss Smartcrypt model environments and important concepts for the systems administrator. They also describe encryption, 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.

Planning for Administration Activities



Smartcrypt Model Environments

Preparing to Use Encryption

Smartcrypt for IBM i comes with strong passphrase-based encryption that can be implemented simply by adding a passphrase and encryption method to any job stream. However, it takes planning and design to ensure a secure and well organized environment for recipient-based encryption. ***Smartcrypt for IBM i*** requires a system administrator to create and manage a local certificate store and/or LDAP server to house the designated digital

certificate key pairs. Easy-to-follow commands and instructions enable system administrators to add, delete, backup, restore, and report on both key pairs and database profiles. In all, this allows for simple and secure methods for administrators to create and maintain a certificate-based encryption environment.

Signing and Authentication

With the implementation of signing and authentication, ***Smartcrypt for IBM i*** allows customers to ensure that data received internally or externally is thoroughly secured. As with certificate-based encryption, the supporting infrastructure for signing and authentication requires design, development, and testing prior to implementing into production, and certificate authority trust chains and revocation lists must be kept up to date. ***Smartcrypt for IBM i*** provides command administration within the local certificate store to add and/or revoke trusted certificate authority (CA) and root certificates. PKWARE recommends system administration adheres to your facility's security policies and implement them accordingly into the ***Smartcrypt*** application.

PKWARE PartnerLink: *SecureZIP Partner for IBM i*

Smartcrypt for IBM i is also available in a special version—***SecureZIP Partner***—through the PKWARE PartnerLink program. The PKWARE SecureZIP Partner program 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.

See Chapter 11 for information on administering and configuring ***SecureZIP Partner for IBM i***. See the SecureZIP Partner chapter in the *Smartcrypt for IBM i User's Guide* for an operational description of SecureZIP Partner. Contact PKWARE for more information about the PKWARE SecureZIP Partner program.

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.

Note: ***PKZIP for IBM i*** provides support for passphrase-based encryption and decryption using a 96-bit "Standard" encryption algorithm that is supported by older ZIP-compatible utilities. In addition, ***PKZIP for IBM i*** Enterprise Edition supports the decryption of all passphrase-based algorithms provided in ***Smartcrypt for IBM i***.

Authentication

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.

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 only in an IFS path. These keyrings are separated into public (only) and secret (both secret and matching public) key sets.

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/or 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/or 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.

Setting Up Stores for Digital Certificates on IBM i

To use certificates for encryption/decryption or digital signing/authentication, Smartcrypt needs to access the keys in the certificates.

Unlike Windows, IBM i does not have a native facility for storing digital certificates and converting them into a form that Smartcrypt can use. To address this, Smartcrypt provides a utility program to set up and manage certificate stores on iSeries for use with Smartcrypt.

Setting Up the Certificate Stores

The PKWARE utilities used to administer the local certificate store is accessed through commands PKCFGSEC, PKBLDCDB, PKQRYCDB, and PKSTORADM. For detailed instructions on creating certificate stores on IBM i, please refer to Chapter 4.

These utilities are used to maintain the stores described in the following table.

Store	*Description*
Public	A store for end-entity certificates used to identify encryption recipients or for authentication of digital signatures. Certificate files in this store contain only public keys; they do not contain private keys. ***Smartcrypt for IBM i*** represents these certificates as files held in the local certificate store path. Normally these IFS files might have a suffix of “cer” or “pem”, but the suffix is not pertinent to Smartcrypt.
Private	A store for end-entity certificate files with their respective private keys. Private keys are used to decrypt files or perform digital signing. ***Smartcrypt for IBM i*** represents these certificates as individual files held in the local certificate store path. Normally these IFS files might have a suffix of “P12” or “PFX”, but the suffix is not pertinent to Smartcrypt. (Private keys in this store are encrypted using PKCS#8 format and PKCS#5 version 2.) Other system types may refer to this store as “Personal” or “MY Store”
Intermediate Certificate Authority	A store file of issuing certificates files associated with the end-entity certificates. These certificates are used to authenticate the validity of an end-entity digital signature on a receiving system. They are also included in a Smartcrypt archive when a signing operation is performed. Other system types may refer to this store as “CA”.
Trusted Root Certificate Authority	A store file of issuing certificates that are classified as “self signed,” meaning that each one is at the top of a hierarchy of issuing CAs. These certificates are used to authenticate the validity of an end-entity digital signature on a receiving system. They are considered “trusted” by virtue of their installation on an authenticating system. They are included in a Smartcrypt archive when a signing operation is performed. Other system types may refer to this store as “ROOT”.
Certificate Revocation List	A store file of static certificate revocation lists published by each CA. Other system types may refer to this store as “CRL”.

All local certificates stores are in a path of the Integrated File System (IFS) and the store paths are defined using the PKCFGSEC command. The public and private stores contains the X.509 certificates, with a certificate locator database providing an index structure that provides search and selection capabilities.

The stores should be defined with directory and object authority that will allow only an administrator to create, change, remove, and update a new certificate store. This authority is also required for creating backups, performing recovery operations, or performing some synchronization tasks which re-allocate components.

Updating the Certificate Stores

X.509 certificates can be added to the local certificate store by copying the files to the public or private store path and then running the PKBLDCDB administration tool to update the certificate locator database. These certificates are frequently obtained on another platform and transferred (binary) to the operational iSeries system for installation.

Important: All X.509 certificates should be transferred to the local iSeries environment in binary mode with no translation.

When certificates are added, the certificate administration tool determines the appropriate store location based on the certificate type specified and dynamically builds an index entry for future search and selection.

Smartcrypt can import certificates and keys in the following file formats:

Format	*Description*
PEM	Contains a single end-entity public-key certificate. It may be in Base-64 encoded (ASCII text with ASCII headers) or DER-encoded binary format. Common file extensions: `.pem`, `.cer`, `.key`
PKCS#12	Contains a single end-entity private-key certificate (which also contains and its public keys). By definition, it is in binary format. Common file extensions: `.pfx`, `.p12`
PKCS#7	Contains one or more CA (and or Root) certificates Common file extension: `.p7b`

You must tell the certificate store administrative command what certificate file type and key type to import.

The stores should be defined with directory and object authority that will allow only an administrator to create, change, remove, and update a new certificate store. This authority is also required for creating backups, performing recovery operations, or performing some synchronization tasks which re-allocate components.

Data Encryption

Smartcrypt for IBM i security functions include strong encryption tools using OpenSSL and IBM software cryptographic facilities. ***Smartcrypt for IBM i*** provides encryption using DES, RC4, 3DES and AES. For OpenPGP files ***Smartcrypt for IBM i*** provides encryption using 3DES, AES and CAST5.

Smartcrypt for IBM i uses a multi-layer key generation process, based on a user-specified passphrase of up to 200 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/or ENTPREC parameters. If a value is present for either setting, whether through parameters 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.

Types of Encryption Algorithms

FIPS 46-3, Data Encryption Standard (DES)

The FIPS (Federal Information Processing Standards) specification 46-3 formerly specified the 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. 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.

Smartcrypt for IBM i supports this algorithm for both ZIP archive and OpenPGP archive files.

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.

Smartcrypt for IBM i supports this algorithm for both ZIP archive and OpenPGP archive files.

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 for non-OpenPGP files.

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.

RC4 is **not** supported for OpenPGP format.

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.

Standard

PKZIP for IBM i provides support for passphrase-based encryption and decryption using a 96-bit “Standard” encryption algorithm that is supported by older ZIP-compatible utilities. Standard is **not** supported for GZIP and OpenPGP format.

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 on key management is available 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 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 for IBM i*** 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 certification authority that cryptographically *binds* the individual’s identity to his or her public key. Binding the individual’s identity to the public key corresponds to the protection afforded to an individual’s private signature key.

A 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.

The certification authority plays a critical role in ensuring the integrity of public keys in the PKI. Upon being presented with proper evidence of identity (usually through a separate entity called a *registration authority*), the CA issues a digital certificate which contains the applicant's public key, identity, and other information (such as duration of the certificate), all signed by the CA's private signature key. The certificate may then be distributed or placed in publicly available databases, called *repositories*.

2 Installation, Licensing, and Configuration

Installation Overview

The installation of ***Smartcrypt**i* is accomplished by following the instructions as summarized below:

- Select the media to be used in installing ***Smartcrypt**i*.
- Install from downloaded file, or CD/DVD.
- Review the README.TXT file for recent information updates.
- Evaluate system requirements.
- Review the present chapter on installation, license, and configuration in this manual and proceed accordingly.
- Run the verification jobs and test product features.
- Begin using the product.

Details of these summarized instructions may be found below.

This chapter describes the process of receiving ***Smartcrypt**i*, either by tape, CD-ROM or by FTP, the process of setting the environment (optional), and the process of installing the license. These processes consists of building the ***Smartcrypt**i* version library, adding the library to your library list, optionally setting environmental pointers and running the install license command with an authorized license.

Even though the default library for ***Smartcrypt**i* Version 14.0 is PKW14961S and is referenced in this manual, the library name for the product can be changed to suit your environmental needs. The ***Smartcrypt**i* product is distributed with a predefined library of PKW140*vrT* (where ***vr*** is the minimum IBM i operation system release supported, **T** is the product type, P=PKZIP and S=Smartcrypt). V6R1M0 would be 61 and V5R4M0 would be 54. If you would like to use another library name or would like to run ***Smartcrypt**i* without it being in the library list, review the operating environment discussion later in this chapter.

It is recommended that you use a generic name for the library that will be used for production (such as PKZIP or PKZIPPROD). This will allow new updates to be implemented without changing a lot of CL programs or processes. To use another library name, review the procedures described later in this chapter (section

"Operating Environment") on how to use the CALL PKZSETLIB program to change the standard library name or run without the library in your library list.

Upgrading From Earlier Versions

If you are upgrading from an earlier version of ***Smartcrypt***^{i}, review the Migration Notes in the *Smartcrypt for IBM i V16.0 Users Guide*.

Also, review the following changes to digital certificate handling:

- ***SecureZIP for i5/OS*** Release 10.0.5 introduced support for digital certificates that are generated with SHA-256 as the internal signing hash algorithm. The installation of a certificate that was generated with an internal signing hash algorithm of SHA-256 may create an operational incompatibility with builds prior to SecureZIP 10.0.5. Care should be taken to coordinate the use of these certificate types when older releases are to be used until a full upgrade can be completed. For more information, see the "Support for Digital Certificates with the SHA-256 RSA Signature Algorithm" section.
- If you have a SecureZIP for i5/OS certificate key store running with releases of SecureZIP prior to v10.0.5, and you need backward compatibility then continue to use the older release to administer certificates within the store. This will avoid the potential of introducing digital certificates with features that are incompatible with the older release. Alternatively, create a new SecureZIP for i5/OS certificate store only to be used with the newer release.

Note: Beginning with SecureZIP for i5/OS v10.0.5, digital certificates using an internal signature algorithm and hash of SHA-256 are supported through the OPENSSL certificate handlers. Older releases used RSA BSAFE CERT-C, which could only handle certificates signed with SHA-1.

Unloading *PKZIP/Smartcrypt for IBM i* from a CD-ROM

The ***Smartcrypt***^{i} installation from a CD-ROM can be performed using either of the following two methods. One method is to use the LODRUN command and the second method would be to restore the library manually. First load the ***Smartcrypt***^{i} CD into the optical unit and note the device name (usually OPT01) along with the correct folder for the DIR parameter.

METHOD A. LODRUN Command

At a command line, type one of the following:

➔LODRUN DEV(*OPT) DIR('/i5OS/Productx ')

or

➔ LODRUN DEV(optical device) DIR('/i5OS/Productx')

where *Productx* is either PKZIP, Secure61, Secure53, or PLink. For example, one of the following:

DIR('/i5OS/PKZIP')
DIR('/i5OS/Secure61') or DIR('/i5OS/Secure53')
DIR('/i5OS/PLink')

LODRUN will first display the following screen showing the default library with the version of ***Smartcrypt***^{i} that will be installed from the CD. The display will wait for a valid library and the pressing of the Enter key to install ***Smartcrypt***^{i}***.*** If the F3 or F12 key is pressed the install will end without installing the product.

LODRUN screen example:

```
  Smartcrypt for IBM i         Version  V16.0

  Install Smartcrypt for IBM i to Library  PKW14961S
      (Note:  Library Must Not Exist)



  Press Enter to Continue
  F3-Exit            F12-Cancel
Error message Area-----------------------------------------------------
```

Next the install process will restore the ***Smartcrypt***^{i} product to the library requested and add the new library to the library list. Then using the program PKZIP program PKZSETLIB, all settings for commands, data area and help files will be resolved to the new library.

At this point, ***Smartcrypt***^{i} is in your library list and is ready to load keys (DEMO or registered) using the INSTPKLIC command as describe later in this chapter.

METHOD B. RSTLIB Command

The ***Smartcrypt***^{i} installation CD-ROM contains a Save file with the product library for the IBM i OS. The **PKW140*vrT*** library contains ***Smartcrypt***^{i} Version 16.0 patch level 0 with the IBM i OS version specified as ***vr*** (IBM i OS version, Target Release, and Modification for the product build—for example, 610 for V6R1M0) and ***T*** as product type (S=Smartcrypt P=PKZIP, and L=SecureZIP Partner).

To unload the required library for ***Smartcrypt***^{i}, load the CD-ROM and issue the following command:

RSTLIB SAVLIB(**PKW140*vrT)*** DEV(***optnn***) OPTFILE(*pkzip*)

Where ***vr*** is the version, release, T is product type and ***optnn*** is the correct optical device name to use.

Using FTP to iSeries to Transfer a PKZIP/Smartcrypt Save File

1. After downloading the product self-extracting file to your PC, extract it. The save file is named PKW14961S.SAV. Be sure to note the path to this file, as it will be required in your FTP session (defaults to C:\PKWARE\SZi5OS\PKW14961S.SAV).
2. Start an iSeries workstation type session, and then sign on with sufficient authority to perform the commands used below.
3. Create a save file on the IBM i - CRTSAVF YourLIB/PKZIP140. The TCP/IP

network server must be running (or start the TCP/IP network server with STRTCP).

4. Start a PC-based FTP session with your normal FTP procedures and send the PC file in binary mode to the iSeries SAVF you created, or run the following:
5. Type "FTP" at the command prompt.
6. Open the iSeries IP address and sign on (ftp> open 208.222.150.11 as an example).
7. Next, type "BIN" at the command line (this transfers the file as a binary object, as is required).
8. Now type "PUT" and then the local file name (in other words, "PUT C:\PKWARE\SZi5OS\PKW14961S.SAV").
9. Finally, type your remote file name (the destination: YourLIB/PKZIP140), and the FTP transfer of the file should start.

Restoring *PKZIP/Smartcrypt for IBM i* Product Library from a Save File

At this time you should have the save file built on your system. Now you can restore the library from the save file.

➔ **RSTLIB SAVLIB(PKW14961S) DEV(*SAVF) SAVF(YourLIB/PKZIP140) RSTLIB(your_pkzip_lib)**

where "your_pkzip_lib" - is the name you want to call the restored PKZIP library.

At this point the ***Smartcrypt**i* product library should exist on iSeries and you can proceed to the Installation procedures below.

If you want, you can now delete the save file created earlier with:

➔ **DLTF YourLIB/PKZIP140**

Installation Procedures

The ***Smartcrypt**i* product library should now exist on the iSeries. All objects including the library are owned by QPGMR. The objects' owner can be change to any valid owner with the CHGOBJOWN command. The ***Smartcrypt**i* library should now be added to the library list (ADDLIBLE your_pkzip_lib).

If you keep the PKZIP library name at PKW14961S, then you do not need to take this additional step. If you change the name from PKW14961S to some other name, then you will need to run a program to setup your PKZIP Environment. In this case you will type on a command line, "CALL PKZSETLIB your_pkzip_lib", where "your_pkzip_lib" is the name of the PKZIP library you restored. This will link the objects to your PKZIP library name. For more information see "How to Change the Standard Library Name" later in this chapter.

Licensing Requirements

***Smartcrypt**[i]* is a licensed product. Without proper licensing, the product can only be used to view archives. Product features and license types can be licensed separately as the user's needs dictate. The license key will contain all of the elements necessary to validate a customer's use of ***Smartcrypt**[i].*

The licensing process is comprised of several key elements that are described in the following sections.

Smartcrypt Partner License Activation

A software license is provided with the ***SecureZIP Partner for IBM i*** package for the purpose of activating, configuring and verifying the installation of the software. A Sponsor Distribution Package must also be obtained independently through the PKWARE SecureZIP Partner program to activate data interchange capabilities with a SecureZIP Partner sponsor.

The SecureZIP Partner software license enables a pre-defined set of features to be run on any system. You are not required to identify your specific processor to be used to run the products.

Trial Period

You will need to contact your reseller or PKWARE, Inc. Sales at 937.847.2374, or email Sales at pksales@pkware.com to initialize your version of ***Smartcrypt**[i]* for the DEMO. If you do not work in the USA, please refer to the GLOBAL CONTACTS.TXT file to contact a dealer in your region.

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.

Reporting Environment

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 16.0 (540) with build
date 2016/12/11
              Current Smartcrypt  Library is PKW14961S
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*[j] 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

Updating License Files

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

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(PKW14961S/PKZLICIN) MBR(PKWARELIC)**

or

➔ **STRSEU SRCFILE(PKW14961S/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(PKW14961S /PKZLICIN) MBR(PKWARELIC)**

```
 Edit File: PKW14961S /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   15/03/03
    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(PKW14961S/PKZLICIN) SRCMBR(PKWARELIC) :**

```
Columns . . . :    1  71           Edit                    PKW14961S/PKZLICIN
SEU==>                                                          PKWARELIC
FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
       ********** Beginning of data *************************************
0001.00 *LICENSED BY PKWARE, Inc   15/03/03
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,  2016/02/26
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 = 0107X8WT, Processor Group =  P10
Evaluation Edition being installed
Smartcrypt Module      - Evaluation set to expire in 31 days on 20160426
Compression           - Evaluation set to expire in 31 days on 20160426
Decompression         - Evaluation set to expire in 31 days on 20160426
GZIP                  - Evaluation set to expire in 31 days on 20160426
Enhanced Decryption   - Evaluation set to expire in 31 days on 20160426
Spool Files           - Evaluation set to expire in 31 days on 20160426
```

```
Large Files           - Evaluation set to expire in 31 days on 20160426
Self Extracting       - Evaluation set to expire in 31 days on 20160426
iPSRA Save/Restore    - Evaluation set to expire in 31 days on 20160426
TapeOut IO Handler    - Evaluation set to expire in 31 days on 20160426
License File PKW14961S/PKZLIC(PKZLIC) Updated successfully
Press ENTER to end terminal session.
```

Licensed Product Features

The license key will be comprised of codes to reflect the product features selected by the customer.

PKZIP Licensing Product Type Cross Reference with Features Codes.

Type	*Feature Code*	*Standard Edition (SE) 77*	*Enterprise Edition (EE) 66*	*x Demo Days 99*
BASE Module	01	Included	Included	Included
Compression	02	Included	Included	Included
Decompression	03	Included	Included	Included
GZIP	04	N/A	Included	Included
Decryption	07	N/A	Included	Included
Spool File Support	08	N/A	Included	Included
Large File Support (ZIP64)	09	N/A	Included	Included
Self Extraction Creator	10	N/A	Included	Included
i5/OS PKWARE Save/Restore Application (iPSRA)	11	Included	Included	Included
1Step2Tape	12	Opt	Included	Included
OpenPGP Support	14	Included	Included	Included

Smartcrypt Licensing Product Type Cross Reference with Features Codes.

Type	*Feature Code*	*Standard Edition (SE) 77*	*Enterprise Edition (EE) 66*	*x Demo Days 99*
BASE Module	01	Included	Included	Included
Compression	02	Included	Included	Included
Decompression	03	Included	Included	Included
GZIP	04	Included	Included	Included
Directory Integration	05	Opt	Included	Included
ADVANCED Encryption	06	Opt	Included	Included
Decryption	07	Included	Included	Included
Spool File Support	08	Included	Included	Included
Large File Support (ZIP64)	09	Included	Included	Included
Self Extraction Creator	10	Included	Included	Included
i5/OS PKWARE Save/Restore Application (iPSRA)	11	Included	Included	Included
1Step2Tape	12	Included	Included	Included
Algorithm Facility	13	Included	Included	Included
OpenPGP Support	14	Included	Included	Included

Licensing Features Code Descriptions

The following table shows the product features available:

Feature Code	*Type*	*Description*
01	**Base Module**	The base module is either PKZIP or Smartcrypt (depending on the product has been selected and which licensed code was issued).
02	**Compression**	This licensable module allows for the compression of data into a ZIP file. This license is required to read the input data and write out the archive or ZIP file.
03	**Decompression**	This licensable module allows for the decompression of data from a ZIP file or an archive. This license is required to read the ZIP archive and to write the IBM i OS datasets during the extraction routine.
04	**GZIP Support**	This licensable module allows a user to read and write GZIP compatible files. GZIP files created using PKZIP for IBM i can be extracted with any compatible GZIP utility on any other platform.
05	**Directory Integration**	This module allows the use of LDAP for certificate accessing. *(Smartcrypt)*
06	**Advanced Encryption Module**	This licensable module allows a user to compress files with advanced encryption methods, encrypt files with certificates, sign files/archives, and authenticate files/archives. *(Smartcrypt)*
07	**Decryption**	This licensable module allows a user to extract files with advanced encryption methods. Note: Feature code 06 is required for certificate decryption.
08	**Spool File Handler**	This licensable module allows a user to compress and extract spool files. The module also allows the spool file to be converted another format, such as a PDF or text format.
09	**Large File Support (ZIP64)**	This licensable feature allows the use of the ZIP64 archive format to support files over 4 Gigabytes and to have more than 65,535 files in an archive.
10	**Self Extract Support**	This licensable feature allows the creation and maintenance of self extracting archives.
11	**i5/OS PKWARE Save/Restore Application (iPSRA)**	This licensable feature allows the application to save data directly to or restore from an archive with a save or restore command.
12	**1Step2Tape**	This licensable feature allows the application to create or read archives directly to tape.
13	**Algorithm Facility**	This licensable feature allows the use of other Encryption/Hashing algorithm APIs.
14	**OpenPGP Support**	This licensable feature allows the use the PKPGPZ and PKPGPU commands to create and extract OpenPGP formatted files.

Splash Screens:

Each time PKZIP or PKUNZIP is executed, the output starts with series of splash screens that shows the product name, copyrights, trademarks, the build data, and the current iSeries serial number. The amount information displayed can be changed at execution time or can be set by changing the defaults of the PKZIP and PKUNZIP commands for the MSGTYPE parameter.

Examples of the normal splash screens are shown below.

Smartcrypt

```
Smartcrypt(R) for IBM i Version 16.0, IBM i <Version>, 2016/02/26
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.
Registered,
Machine ID = 01061B5F, Processor Group =  P05, OS=V5R3M0
```

PKZIP

```
PKZIP(R) for IBM i Version 16.0, 2016/02/26
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.
PKZIP(R) is a registered trademark of PKWARE, Inc.
Registered,
Machine ID = 01061B5F, Processor Group =  P05, OS=V5R3M0
```

SecureZIP Partner

```
SecureZIP(R) Partner for IBM i Version 16.0, 2016/02/26
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.
SecureZIP(R) is a trademark of PKWARE, Inc.
SecureZIP Partner for IBM i License
Machine ID = 01061B5F, Processor Group =  P05, OS=V5R3M0
```

Record Layouts

The record layouts for control file records (portions of which your reseller of PKWARE, Inc. will provide) are described below for each type. Remember, the columns are relative to column 1 in a source file and if you use EDTF then column 1 is now column 13.

Comment Control Card

Comment cards are free format and begin with an * in column 1.

Customer Control Card

Customer records always begin with a “55” or ‘57”.

	Control Code	Check Characters	Customer Number	Customer Name
Format	55	cccccccc	nnnnnnnnn	xxx . . . xxx
Columns	1-2	4-11	13-21	23-74
Length	2	8	9	50

55 Feature Control Code (always a 55 or 57)

cccccccc Check Character (supplied by PKWARE, Inc.)

nnnnnnnnn	Customer Number (supplied by PKWARE, Inc.)
xxx . . . xxx	Customer Name (supplied by customer - must be alphanumeric (AA-99)

Feature Control Card

	Feature Code	Check Characters	Expiration Date	Hardware Information
Format	ff	cccc	yyyymmdd	sssssssttt
Columns	1-2	4-11	13-20	22-33
Length	2	8	8	12

ff	Feature Control Code (provided by PKWARE, Inc.)
cccccccc	Check Character (provided by PKWARE, Inc.)
yyyy	year (2001)
mm	month (01-12)
dd	day (01-31)
sssssss	CPU serial # (12345ABC)
tttt	CPU Processing Group (P10)

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

Install Demo

```
Smartcrypt(R) for IBM i Version 16.0,  2016/02/26
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
Rec -  1 *LICENSED BY PKWARE   05/26/15 WSS
Rec -  2 *Smartcrypt with Enterprise License
Rec -  3 57 MR6CCP2B 000015319 PKWARE, INC.
Rec -  4 99 HH6QYPKK 20070626 01061B5FP05
Evaluation Edition being installed
Smartcrypt Module      - Evaluation set to expire in 31 days on 20160426
Compression           - Evaluation set to expire in 31 days on 20160426
Decompression         - Evaluation set to expire in 31 days on 20160426
GZIP                  - Evaluation set to expire in 31 days on 20160426
Enhanced Decryption   - Evaluation set to expire in 31 days on 20160426
Spool Files           - Evaluation set to expire in 31 days on 20160426
Large Files           - Evaluation set to expire in 31 days on 20160426
Self Extracting       - Evaluation set to expire in 31 days on 20160426
iPSRA Save/Restore    - Evaluation set to expire in 31 days on 20160426
TapeOut IO Handler    - Evaluation set to expire in 31 days on 20160426
License File PKW14961S/PKZLIC(PKZLIC) Updated successfully
```

Install SE

```
Smartcrypt(R) for IBM i Version 16.0,  2016/02/26
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
Rec -  1 *LICENSED BY PKWARE   05/26/15 WSS
Rec -  2 * Standard features licensing
Rec -  3 57 MR6CCP2B 000015319 PKWARE, INC.
Rec -  4 77 AH6QYPKR 20070526 01061B5FP05
Standard Edition being installed
License File PKW14961S/PKZLIC(PKZLIC) Updated successfully
```

Install EE

```
Smartcrypt(R) for IBM i Version 16.0,  2016/02/26
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
Rec -  1 *LICENSED BY PKWARE   05/26/15 WSS
Rec -  2 *Smartcrypt with Enterprise License
Rec -  3 57 MR6CCP2B 000015319 PKWARE, INC.
Rec -  4 66 MH6QYPKB 20070526 01061B5FP05
Enterprise Edition being installed
License File PKW14961S/PKZLIC(PKZLIC) Updated successfully
```

Reporting

To report on the status of a license at your location, you can run the license program with the following command: INSTPKLIC TYPE(*VIEW).

Examples from above:

View Demo

```
Smartcrypt(R) for IBM i Version 16.0,  2016/02/26
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#
10.0 Product Licensed to Customer # 000014581 -PKWARE Internal Demo Customer
*****************************************
Compression           -DEMO with 23 Days remaining (04/03/2016)
      Contact PKWARE, Inc. for Licensing
*****************************************
Decompression         -DEMO with 23 Days remaining (04/03/2016)
      Contact PKWARE, Inc. for Licensing
*****************************************
GZIP                  -DEMO with 23 Days remaining (04/03/2016)
      Contact PKWARE, Inc. for Licensing
*****************************************
IFS File Handlers     -DEMO with 23 Days remaining (04/03/2016)
      Contact PKWARE, Inc. for Licensing
*****************************************
Database File Handlers-DEMO with 23 Days remaining (04/03/2016)
      Contact PKWARE, Inc. for Licensing
*****************************************
Advanced Encryption   -DEMO with 23 Days remaining (04/03/2016)
      Contact PKWARE, Inc. for Licensing
```

```
*****************************************
Spool Files           -DEMO with 23 Days remaining (04/03/2016)
      Contact PKWARE, Inc. for Licensing
*****************************************
Self Extracting       -DEMO with 23 Days remaining (04/03/2016)
      Contact PKWARE, Inc. for Licensing
*****************************************
Press ENTER to end terminal session.
```

View SE

```
Smartcrypt(R) for IBM i Version 16.0,  2016/02/26
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 05/26/10  14:17  from CPU Serial# 01061B5F
10.0 Product Licensed to Customer # 000015319 -PKWARE, INC.
*****************************************
Smartcrypt Module      :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
Compression           :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
Decompression         :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
GZIP                  :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
Directory Integration :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
Enhanced Encryption   :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
Enhanced Decryption   :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
Spool Files           :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
Large Files           :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
Self Extracting       :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
iPSRA Save/Restore    :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
TapeOut IO Handler    :Licensed -Expires 02/28/2400 for processors:
  Serial# 01061B5F    Processor Type P05
*****************************************
```

View Single with Exception

```
Smartcrypt(R) for IBM i Version 16.0,  2016/02/26
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#
10.0 Product Licensed to Customer # 000003079 -Key Testers Inc.
*****************************************
Compression           :Licensed -Expires 02/28/2400 for processors:
  Serial# 0107X8WT    Processor Type P10
*****************************************
Decompression         :Licensed -Expires 02/28/2400 for processors:
  Serial# 0107X8WT    Processor Type P10
*****************************************
GZIP                  :Licensed -Expires 02/28/2400 for processors:
  Serial# 0107X8WT    Processor Type P10
*****************************************
IFS File Handlers     :Licensed -Expires 02/28/2400 for processors:
  Serial# 0107X8WT    Processor Type P10
*****************************************
Database File Handlers:Licensed -Expires 02/28/2400 for processors:
  Serial# 0107X8WT    Processor Type P10
*****************************************
Advanced Encryption   :Licensed -Expires 02/28/2400 for processors:
  Serial# 0107X8WT    Processor Type P10
*****************************************
Spool Files           :Licensed -Expires 02/28/2400 for processors:
  Serial# 0107X8WT    Processor Type P10
*****************************************
Self Extracting       :Licensed -Expires 00/00/0000 for processors:
Serial# 0107X8WT is Not Licensed.  Use Of The Product will CEASE in 7 days
(6/17/2003)
*****************************************
Press ENTER to end terminal session
```

Conditional Use

PKWARE, Inc. 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 customer, ***Smartcrypt***^{i} has a process that will allow a customer to continue using the product for a period of seven days. During this time, error messages will be displayed on the console (as well as the printout) for each execution of ***Smartcrypt***^{i}***.*** At the end of the grace period, if the license keys are not updated, the product will no longer function in any environment other than to view an archive. During the grace period, the program will continue to function but will issue warning messages regarding the grace period.

Before the end of the grace period, you must contact your reseller or PKWARE, Inc. at 1-937-847-2687 to obtain licensing to allow continued use.

Operating Environment

Smartcrypt^{i} is distributed in a standard library, such as PKW14961S where the 140 is the version release of ***Smartcrypt***^{i} and 610 is the minimum IBM i OS operation system release that it supports (such as V6R1M0). The ***Smartcrypt***^{i} library contains programs, commands, help panels, message file, license files, translation source file, command source file, CL source file (CVTNAME CL program and examples), and the ***Smartcrypt***^{i} control data area necessary to run the product. As released, the distributed library PKW14961S must be in the library list and the library name must be the same as distributed.

The library name used by ***Smartcrypt**i* is determined by the data area PKZDTA5.

PKZDTA5 Data Area

The "PKZDTA5" data area is required to be in the library list in order run the programs and commands of ***Smartcrypt**i*.* The data area is 50 bytes in length with the following layout:

Bytes 1 thru 10 - Library name for the product (required to run the product).

Bytes 11 thru 20 - Release Levels for ***Smartcrypt**i*.*

Bytes 21 thru 40 - Distribution and Generation Information only.

Bytes 41 thru 50 - Global settings.

Byte 41 - Disallow wildcard selection

N = Wildcard selection is permitted (default).

Y = Wildcard selection with PKZIP is not permitted.

Bytes 42-50 Internally Reserved

An example of the "DSPDTAARA PKZDTA5" output might look like:

```
                              Display Data Area
Data area . . . . . . . :   PKZDTA5
  Library . . . . . . . :     PKW14961S
Type  . . . . . . . . . :   *CHAR
Length  . . . . . . . . :   50
Text  . . . . . . . . . :   Smartcrypt for IBM i  Library-V16.0

           Value
Offset      *...+....1....+....2....+....3....+....4....+....5
    0      'PKW14961S V16.0             nnOV6R1M0 NB---- oS '
```

The running of ***Smartcrypt**i* requires the data area to determine the library name to use for running the commands, help panels, and message file.

How to Change the Standard Library Name

To assist the customer in changing the environment for running the ***Smartcrypt**i* product, the program "PKZSETLIB" has been included in the distribution library. PKZSETLIB will set the PKZDAT5 data area's library name and will change the commands and help panels to recognize the library links.

An example of how use PKZSETLIB to change the standard library name to a user environment library name is shown in the steps below:

1. Rename the ***Smartcrypt**i* standard library to the new library name:

   ➔ **RNMOBJ OBJ('PKW14961S) OBJTYPE(*LIB) NEWOBJ(newlib)**

2. Verify that newlib is in library list with ADDLIBLE or EDTLIBL:

   ➔ **ADDLIBLE newlib**

3. Execute PKZSETLIB program to change the data area and change links:

   ➔ **CALL PKZSETLIB ('newlib')**

4. Display the PKZDTA5 data area and verify that the Library has been changed.

   ➔ **DSPDTAARA PKZDTA5** (Positions 1 thru 10 should contain newlib)

5. Test the commands PKZIP or PKUNZIP to make sure they execute correctly.

At this point the "newlib" is the new ***Smartcrypt***^{i} library for this environment and will need to be placed in the library list to run.

Alternate Install from SAVF with Non-Standard Library Name

After the SAVF is ready to restore the library, a slight modification to the commands as shown in above will be required. In the RSTLIB command you will specify "newlib" in the parameter RSTLIB to restore the library with the new library name.

For example:

➔ **RSTLIB SAVLIB('PKW14961S) DEV(*SAVF) SAVF(mylib/PKW140) RSTLIB(newlib)**

To complete the installation, do the following:

1. Add the "newlib" to your library list with ADDLIBLE:

   ➔ **ADDLIBLE newlib**

2. Execute the PKZSETLIB program to change data area and change links:

   ➔ **CALL PKZSETLIB ('newlib')**

3. Display the PKZDTA5 data area and verify that the Library has been changed.

   ➔ **DSPDTAARA PKZDTA5** (Positions 1 thru 10 should contain newlib)

4. Test the commands PKZIP or PKUNZIP to make sure they execute correctly.

At this point the "newlib" is the new ***Smartcrypt***^{i} library for this environment and will need to be placed in the library list to run.

Running Without the *PKZIP/Smartcrypt* Library in the Library List

Smartcrypt^{i} can run without the library in the library list as long as a data area as described above is in a library that is in the library list. One method is qualifying the command with the library name, such as:

➔ **my*library*/PKZIP ARCHIVE('myarchive/V5(test1)' FILES("testlib/*all')**

The other method copies the commands to a library that exist in their library List.

Note: The running of ***Smartcrypt***^{i} requires the data area to determine the library name to use for running the commands, help panels, and message files. The following are five steps to run ***Smartcrypt***^{i} without its distribution library in the library list.

Assume 'PKW14961S' is the ***Smartcrypt***^{i} library and the customer's library MYlibrary is in the customer's standard library list. We will now add three new objects to the library MYlibrary.

1. Create a new data area in the chosen library MYlibrary:

   ➔ **CRTDTAARA DTAARA(MYlibrary/PKZDTA5) TYPE(*CHAR) LEN(50)**

TEXT('PKZIP control data area')

2. Next set the data area value to the same values found in supplied data area in the PKZIP Library.

   ➔ **DSPDTAARA PKW14961S/PKZDTA5**

   Output might look like this:

```
                              Display Data Area
Data area . . . . . . . :   PKZDTA5
  Library . . . . . . . :     PKW14961S
Type  . . . . . . . . . :   *CHAR
Length  . . . . . . . . :   50
Text  . . . . . . . . . :   Smartcrypt for IBM i  Library-V16.0

           Value
Offset      *...+....1....+....2....+....3....+....4....+....5
0          'PKW14961S V16.0             NNOV6R1M0 N--       '
```

3. Now change the data area:

   ➔ **CHGDTAARA DTAARA(MYlibrary/PKZDTA5) VALUE(''PKW14961S V16.0xxxV6R1M0')**

4. Next copy the two commands to the MYlibrary Library.

   ➔ **CRTDUPOBJ OBJ(PKZIP) FROMLIB(PKW14961S) OBJTYPE(*CMD) TOLIB(MYlibrary)**

   ➔ **RTDUPOBJ OBJ(PKUNZIP) FROMLIB(PKW14961S) OBJTYPE(*CMD) TOLIB(MYlibrary)**

   At this point, PKZIP will work for all users that have the library MYlibrary in their library list.

Note: When installing another ***Smartcrypt****i* version, the same process should be followed in order to pick up the latest settings of the data area and commands.

Next ensure the ***Smartcrypt****i* library is NOT in your library list and run a few sample tests for PKZIP and PKUNZIP to make sure the product works.

3 Security Administration Overview

Requires Smartcrypt

This section discusses how to use ***Smartcrypt for IBM i*** to secure your data. Elements that are required to make a ***Smartcrypt for IBM i*** archive are discussed in detail. These elements, when selectively used, combine to create a ***Smartcrypt for IBM i*** archive or allow the extraction of a file or files from a ***Smartcrypt for IBM i*** created archive.

A series of commands and CLPs are provided to assist you in building and maintaining the Smartcrypt certificate store. They come standard with ***Smartcrypt for IBM i***. The Smartcrypt commands that are used to accomplish these tasks are shown in this chapter, along with notes and comments.

Keywords, Phrases, and Acronyms Used in This Chapter

Smartcrypt for IBM i introduces new terminology to users that are familiar with PKZIP. These expressions directly relate to the security features inherent in ***Smartcrypt for IBM i.***

- Public key certificate(s)
- Private key certificate(s)
- Data base profile (local certificate store)
- LDAP profile (networked certificate store)
- Passphrase
- Recipient
- Contingency Key (contingency recipient)
- Configuration profile
- Certificate store
- Certificate authority intermediate store
- Trusted root store
- Certificate revocation list (CRL) store
- Enterprise security configuration

- Common name
- Path
- Certificate configuration
- PING
- TCP/IP
- User certificate
- Certificate authority
- Recipient database
- Recipient searches
- File name encryption

Accessing Public and Private Key X.509 Certificates

Smartcrypt for IBM i provides access to both public and private key certificates through a set of IFS files and a DB2 locator database when the parameter ENTPREC(*DB:...) or ENTPREC(*FILE:...) is requested.

In addition, ENTPREC(*LDAP:...) requests are resolved through configured network definitions.

Public Key Certificate

Certificate-based encryption allows the exchange of encrypted data without the security risks of exchanging or retaining a passphrase. This form of encryption uses a public-key digital certificate when creating an archive and it then uses a corresponding private-key certificate by the recipient to decrypt the archive. Digital certificates may be identified and selected by naming information, such as "common name" or an email address.

To do this ***Smartcrypt for IBM i*** performs a process called *digital enveloping* using digital certificates when encrypting data for specified public key recipients. The 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 of which is signed by a certificate authority (CA). The CA essentially guarantees that the public key belongs to the named entity.

Private Key Certificates

To UNZIP a file that has been encrypted with a public-key certificate, the receiver must supply a matching private-key certificate. This is done by including ENTPREC parameter that specifies the location of the private-key certificate along with its associated access passphrase. Note that this passphrase is not used to encrypt a file but rather to access the private-key certificate.

ENTPREC parameters can be coded directly (up to 30 entries) or can point to the Inlist input stream file. One method is for each user to create an Inlist for their private key information and define the security such that only their user ID can read this profile Inlist. A private-certificate Inlist designates a saved repository of the private key certificates.

Enterprise Security Configuration

Smartcrypt utilizes an enterprise security configuration file to define the enterprise policies. With the command PKCFGSEC, you can view the policies or make revisions to these policies. PKCFGSEC updates and views the Smartcrypt security configuration file. This configuration file is where the enterprise security options and defaults reside. It is recommended that you define your standard enterprise options in a CL using the PKCFGSEC command. Then as changes occur or new versions are upgraded the CL can be run to reset the parameters. This will also help if you need to temporarily change one or more settings, as the CL provides a quick means to reset the configuration back to the enterprise setting.

> **Note: SecureZIP Partner Read/Write Mode: Supplemental administration activities unique to *SecureZIP Partner for IBM i* are covered in Chapter 11 in the section "SecureZIP Partner Certificate Store Administration and Configuration."**

It is suggested that a specific user ID (such as ZIPADMN), user or group be assigned to administrate the enterprise configuration. This can be accomplished by changing the security for the configuration file PKZCFG in the Smartcrypt library with the EDTOBJAUT command. First *Public should be changed to use only the file PKZCFG; then set the administrator as the owner with security for changes. For example:

```
                          Edit Object Authority

Object . . . . . :   PKZCFG          Owner  . . . . . :   ZIPADMN
  Library  . . . . :     PKW140XXS     Primary group  . . . :   *NONE
Object type  . . . . :   *FILE          ASP device . . . . . :   *SYSBAS

Type changes to current authorities, press Enter.

  Object secured by authorization list  . . . . . .   *NONE

                         Object    ----------Object-----------
User        Group       Authority  Opr  Mgt  Exist  Alter  Ref
*PUBLIC                 *USE        X
ZIPADMN                 *ALL        X    X     X      X     X
```

Contents of the Configuration Profile

Below is a sample of the PKCFGSEC command. For more information about the command, see Chapter 6. Before running ***Smartcrypt for IBM i***, it is recommended that all parameters be reviewed and defined for your enterprise.

```
                  Smartcrypt Secure Config  (PKCFGSEC)

Type Processing  . . . . . .   *UPDATE       *UPDATE, *VIEW
ZIP Secure Settings:
    Smartcrypt Active  . . . . .   *YES          *NO, *YES, *SAME
    AES 3DES Keys . . . . . .   *YES          *NO, *YES, *SAME
    Passphrase. . . . . .   *NO           *NO, *RQD, *SAME
    Recipient Secure  . . . . .   *NO           *NO, *RQD, *SAME
    File Signing  . . . . . .   *NO           *NO, *SAME
    Archive Signing   . . . . .   *NO           *NO, *SAME
    File Authentication   . . .   *NO           *NO, *SAME
```

```
     Archive Authentication  . .   *NO           *NO, *SAME
     File Name Encryption  . . .   *NO           *NO, *RQD, *OPT, *SAME

Strong Encryption:
       Lockdown Method . . . . . > AES          *SAME, *NONE, AES, AES128...
Hash  . . . . . . . . . . > *SHA1         *SAME, *SHA1
Archive Passphrase Specs:
     Min Length  . . . . . . . . > 1            1-260, *SAME
     Max Length  . . . . . . . . > 260          1-260, *SAME
     Hardening Options . . . . . > 0            *SAME, 0, 1, 2
Signing Settings:
    Signing Hash . . . . . . . . > *SHA1        *SAME, *SHA1, *MD5...
    Validate Level     . . . . . > *VALIDATE
       Lockdown Filters  . . . . > *NO          *NO, *YES, *SAME
       Filters . . . . . . . . . > *ALL         *SAME, *ALL, *NONE...
              + for more values
Authenticate Filters:
    Validate Level     . . . . . > *VALIDATE
       Lockdown Filters  . . . . > *NO          *NO, *YES, *SAME
       Filters . . . . . . . . . > *ALL         *SAME, *ALL, *NONE...
              + for more values
Encryption Filters:
    Validate Level     . . . . . > *VALIDATE
       Lockdown Filters  . . . . > *NO          *NO, *YES, *SAME
       Filters . . . . . . . . . > *ALL         *SAME, *ALL, *NONE...
              + for more values
Decryption Filters:
    Validate Level     . . . . . > *NONE        *VALIDATE, *WARN, *NONE...
Revocation Settings:
    CRL Type           . . . . . > *NONE        *STATICCRL, *NONE, *SAME
    Revoke Level       . . . . .   *SAME        *NONE, *SAME
Certificate Validation Level:
    Cert. Best Fit . . . . . . . > *NONE        *NONE, *LATESTVALID...
    Cert. Secure Type  . . . . . > *NONE        *NONE, *ENCRYPTION...
Certificate DataBase Locator:
     Active? . . . . . . . . . . > *YES         *NO, *YES, *SAME
     Store Type  . . . . . . . . > *LIB         *LIB, *SAME
     Sequence  . . . . . . . . . > 0            0-9
     Library . . . . . . . . . . > PKW14961S    DB Library
     Search Mode . . . . . . . . > *EMAIL       *EMAIL, *CN, *SAME
Certificate PUBLIC Store:
     Store Type  . . . . . . . . > *PATH        *PATH, *SAME
     Sequence  . . . . . . . . . > 1            0-9
     Store Location  . . . . . . > '/yourpath/PKWARE/Cstores/Public'

Certificate PRIVATE Store:
     Store Type  . . . . . . . . > *PATH        *PATH, *SAME
     Sequence  . . . . . . . . . > 0            0-9
     Store Location  . . . . . . > '/yourpath/PKWARE/Cstores/Private'

Certificate CA Store:
     Store Type  . . . . . . . . > *FILE        *FILE, *SAME
     Sequence  . . . . . . . . . > 0            0-9
     Store Location  . . . . . . > '/yourpath/PKWARE/Cstores/CA/pkwareCA.store'

Certificate ROOT Store:
     Store Type  . . . . . . . . > *FILE        *FILE, *SAME
     Sequence  . . . . . . . . . > 0            0-9
     Store Location  . . . . . . > '/yourpath/PKWARE/Cstores/Root/pkwareRT.store'

Certificate CRL Store:
     Store Type  . . . . . . . . > *FILE        *FILE, *SAME
     Sequence  . . . . . . . . . > 0            0-9
     Store Location  . . . . . . > '/yourpath/PKWARE/Cstores/CRL/pkwareCRL.store'

Certificate Work Path:
     Path Location . . . . . . . > '/yourpath/PKWARE/Cstores/CTemp'
```

```
SecureZIP Partner Path:
      Path Location . . . . . .   *SAME

Contingency Key(s):
    LookUp Type  . . . . . . . . > *MBRSET       *NONE, *DB, *LDAP, *FILE...
    Recipient  . . . . . . . . . > 'pkwareCertAdmin04.cer'
    Required . . . . . . . . . . > *RQD          *RQD, *OPT, *SAME
OpenPGP Definitions:
   Allow Keys for Smartcrypt  . .   *SAME         *NO, *YES, *SAME
   Allow PKPGPZ (ZIP)  . . . . .   *SAME         *NO, *YES, *SAME
   Allow PKPGPU (UNZIP)  . . . .   *SAME         *NO, *YES, *SAME
   Encryption Key Select . . . .   *SAME         *NONE, *LATESTVALID...
   Signing Key Select  . . . . .   *SAME         *NONE, *LATESTVALID...
OpenPGP PUB Keyring:
    Keyring Handle   . . . . . .   *SAME         Handle, *NONE, *SAME
    Keyring Engine   . . . . . .   *FILE         *FILE
    Engine Name  . . . . . . . .
LDAP Definitions:
     Nbr Active LDAPs  . . . . . > 3            1-99, *NONE, *SAME
  Primary LDAP:
     1 Network Name  . . . . . . > '192.168.132.25'

     1 Port  . . . . . . . . . . > 389          1-9999
     1 Sequence  . . . . . . . . > 1            0-9
     1 TimeOut . . . . . . . . . > 0            0-9999
     1 Access User . . . . . . . > 'PKWAREADDEV\test'

     1 Access Passphrase . . . . > 'xxxxxx'

     1 Search Mode . . . . . . . > *EMAIL       *EMAIL, *CN
     1 Start Node  . . . . . . . > 'cn=users,dc=pkwareaddev,dc=com'

     1 Test  . . . . . . . . . . > N            N, Y

  Secondary LDAP:
     2 Network Name  . . . . . . > '192.168.115.15'

     2 Port  . . . . . . . . . . > 389          1-9999
     2 Sequence  . . . . . . . . > 2            0-9
     2 TimeOut . . . . . . . . . > 0            0-9999
     2 Access User . . . . . . . > 'cn=Jon Smith,ou=browndeer,o=pkware,c=US,cn=
user,dc=cosmos,dc=pkware,dc=com'
     2 Access Passphrase . . . . > 'xxxxxxx'

     2 Search Mode . . . . . . . > *EMAIL       *EMAIL, *CN
     2 Start Node  . . . . . . . >
                                 'o=pkware,c=US,cn=user,dc=cosmos,dc=pkware,d=com'
     2 Test  . . . . . . . . . . > N            N, Y

  Tertiary LDAP:
     3 Network Name  . . . . . . > '192.168.115.32'

     3 Port  . . . . . . . . . . > 4389         1-9999
     3 Sequence  . . . . . . . . > 3            0-9
     3 TimeOut . . . . . . . . . > 0            0-9999
     3 Access User . . . . . . . > ''

     3 Access Passphrase . . . . > 'xxxxxx'

     3 Search Mode . . . . . . . > *EMAIL       *EMAIL, *CN
     3 Start Node  . . . . . . . > 'o=PKWARE'

     3 Test  . . . . . . . . . . > Y            N, Y
```

File and Data Base (DB) (Local Certificate Store)

During ***Smartcrypt for IBM i*** processing that requires encryption intended for an ENTPREC, the associated public-key certificate(s) must be located. One way of designating which public-key recipients to include is through an IFS file or by using the DB to locate the recipients with the ENTPREC parameter. By using DB or LDAP, this allows for recipient selection based on name or email address through a configured database of certificates on the system that is executing ***Smartcrypt for IBM i***. 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 one to many recipients.

Your technical support staff is responsible for configuring the local certificate store and should provide you with information on which method they prefer.

LDAP Profile (Networked Certificate Store)

During ***Smartcrypt for IBM i*** processing that requires encryption intended for a recipient, the associated public-key certificate(s) must be located.

One way of designating which public-key recipients to include is through the LDAP interface to a directory server in the ENTPREC parameters. This allows for recipient selection based on name, email address or other installation-configured LDAP fields. One or more LDAP compliant servers may be configured for searching.

Your technical support staff is responsible for configuring the LDAP-compliant directory that stores certificates and will provide you with information on the enterprise approach for your facility.

Certificate Authority, Root Certificates, and Certificate Revocation List Stores

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 as a store format. The intermediate CA certificates are maintained independently from the root certificates. These files are defined to Smartcrypt in the PKCFGSEC with the CSCA and CSROOT parameters.

If a certificate revocation list (CRL) is to be used, the certificate revocation list store will contain all CRLs for revocation processing. The CRL store is separate from the CA store for more efficient certificate processing. The store is defined in PKCFGSEC with parameters CSCRL. To use the CRL, Smartcrypt needs to know that a static CRL is used. This is defined with the REVKEP parameter in PKCFGSEC.

Support for Digital Certificates with the SHA-256 RSA Signature Algorithm

With ***SecureZIP for IBM i*** Release 10.0.5, support for digital certificates that are generated with SHA-256 as the internal signing hash algorithm was introduced.

Installing a certificate that was generated with an internal signing hash algorithm of SHA-256 will create an operational incompatibility with builds prior to SecureZIP 10.0.5. This can occur when an external Certificate Authority provides such certificates and they are added for use with Smartcrypt. These incompatibilities include:

- SecureZIP releases that do not support the SHA-256 signing algorithm within the certificate cannot open an end-entity certificate generated with SHA-256 referenced for RECIPIENT encryption/decryption, digital signing, or signature authentication. Certificates signed with either MD5 or SHA-1 referenced within the key store index will continue to be available for processing.

  While the key store index remains in compatibility mode for SecureZIP releases 9.0 through 10.0.0, a reminder will be issued whenever an end-entity SHA-256 signed certificate (either public-key only or public/private key pair) is registered. If earlier releases of SecureZIP are not to be used with a key store index, the use of program PKCSCVRT to register SHA-256 support into the key store index will disable the reminder mechanism.

- When a ROOT or Certificate Authority certificate having an internal signing hash algorithm of SHA-256 is installed, the entire key store index will be rendered inoperative for SecureZIP 10.0.0 and prior.

  While the key store index remains in compatibility mode for SecureZIP releases 9.0 through 10.0.0, a pop-up message will be issued after the certificate authority SHA-256 signed certificate installation was made. If the administrator decides that this store is required to support builds prior to SecureZIP 10.0.5, then the administrator will have to delete the certificates from the CA or Root store that creates the incompatibility. If the administrator confirms that the installation should proceed they can pres PF8 to update the key store conversion and will accept all certificates signed with SHA-256 and higher without warning pop-up messages.

When an environment is known to use software releases that support the SHA-256 RSA signing algorithm for digital certificates, they can CALL PGM(PKCSCHKR) PARM(CNTL01) to mark the Smartcrypt Key Store index for SHA-256 processing. This will eliminate administrative pop-up messages from occurring when using the certificate add procedure.

For other details view the file $PKCSUPRT in the Smartcrypt library such as DSPF FILE(PKW14961S/$PKCSUPRT) MBR(CNTL01).

Define Stores Paths Program PKSETPATHS

The distribution includes the program **PKSETPATHS**, used to define required folders and restore empty stores for the CA, Root, and CRL Store.

The PKSETPATH CL sets and verifies all folders required for Smartcrypt and SecureZIP Partner. The two parameters required for input are

- The Smartcrypt Library
- The base path where the Smartcrypt and SecureZIP Partner folders will exist.

The base path should exist and should have the security of the folder set. New folders that you create inherit the security of the inputted base path.

After all required folders are verified or created, empty initial stores for the CA, Root, and CRL will be restored.

Smartcrypt folders verified or created

/Public	Public store path
/Private	Private store path
/CA	Certificate Authority Store
/Root	Trusted Authority Root Store
/CRL	Certificate Revocation List
/CTemp	Smartcrypt work path

SecureZIP Partner folders verified or created

/Sponsor	SecureZIP Partner Parent Folder
/Sponsor/Package	Sponsor package folder
/Sponsor/Info	Sponsor Information folder for installed sponsors
/Sponsor/Auth	Sponsor Read mode files folder
/Sponsor/Recip	Sponsor Write mode files folder
/Sponsor/CA_Auth	Reserved for Future use
/Sponsor/LOG	Sponsor Install Log file folder
/Sponsor/Temp	Sponsor Install work folder

For SecureZIP Partner the distributed PKWARE package PKWARE.dat is restored to the /Sponsor/Package folder, and an initial PKCFGSEC is done to set the environment.

The following initial empty stores will be restored: 1) /CA/Pkz00CA.store, 2) /Root/Pkz00ROOT.store, and 3) /CRL/Pkz00CRL.store. If the files exist, the restore will not restore the files.

EXAMPLE: ➔CALL PGM(PKSETPATHS) PARM('MYZIPLIB' '/myroot/pkware/CStore')

Where 'MYZIPLIB' is the Smartcrypt library and '/myroot/pkware/CStore ' is the full path where all the stores folders will be defined.

The following sections will talk about creating or making a folder/directory for the stores, but by call the PKSETPATHS CL program, all required folders build be created if they do not exist.

Creating Store Folders and Stores.

The information in following table assists with allocating the components necessary to define the certificate authority, root, and CRL stores.

The following stores are maintained by the utility PKSTORADM command:

Intermediate Certificate Authority	A store of issuing certificates files associated with the end-entity certificates. These certificates are used to authenticate the validity of an end-entity digital signature on a receiving system. They are also included in a Smartcrypt archive when a signing operation is performed. Other system types may refer to this store as “CA”.
Trusted Root Certificate Authority	A store of issuing certificates that are classified as “self signed,” meaning that each one is at the top of a hierarchy of issuing CAs. These certificates are used to authenticate the validity of an end-entity digital signature on a receiving system. They are considered “trusted” by virtue of their installation on an authenticating system. They are included in a Smartcrypt archive when a signing operation is performed. Other system types may refer to this store as “ROOT”.
Certificate Revocation List	A store of certificate revocation lists published by each CA. Other system types may refer to this store as “CRL”.

First define the path and file name with PKCFGSEC parameters: CSCA (intermediate certificate authority), CSROOT (trusted root certificate authority), and CSCRL (certificate revocation list).

Packaged in the TSTCERTS archive are three empty initialized stores (`pkwareCA.store`, `pkwareRT.store`, and `pkwareCRL.store`) that can be restored to your directories. After restoring the store files, you can name the files anything that you want to match your definition in PKCFGSEC.

For example:

1. First create all store folders and paths:

   ➔ **CALL PGM(PKSETPATHS) PARM('PKW14961S' '/myroot/pkware/CStore')**

2. Extract the initialized stores:

   Extract initialized empty CA intermediate certificate store:

   ➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts') FILES('pkwareCA.store') TYPE(*EXTRACT) EXDIR('/myroot/pkware/CStore/CA') DROPPATH(*ALL) TYPARCHFL(*DB) TYPFL2ZP(*IFS) CVTTYPE(*NONE)**

   Extract initialized empty trusted root store:

   ➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts') FILES('pkwareRT.store') TYPE(*EXTRACT) EXDIR('/myroot/pkware/CStore/Root') DROPPATH(*ALL) TYPARCHFL(*DB) TYPFL2ZP(*IFS) CVTTYPE(*NONE)**

Extract initialized empty revocation list store:

➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts') FILES('pkwareCRL.store') TYPE(*EXTRACT) EXDIR('/myroot/pkware/CStore/CRL') DROPPATH(*ALL) TYPARCHFL(*DB) TYPFL2ZP(*IFS) CVTTYPE(*NONE)**

3. Change the security of the files and paths.

It is suggested that a specific user ID (such as ZIPADMN), user, or group be assigned to administrate the stores. This can be accomplished by changing the security for the paths and files created above with the CHGAUT and CHGOWN commands. First change *Public to only *Use for the files, and then set the administrator as the owner with security for changes.

For example changes for the Root Store path and file might be:

Change ownership to ZIPADMN:

➔ **CHGOWN OBJ('/myroot/pkware/CStores/Root/') NEWOWN(ZIPADMN)**
➔ **CHGOWN OBJ('/myroot/pkware/CStores/Root/pkwareRT.store') NEWOWN(ZIPADMN)**

Change *PUBLIC for use of root path and file:

➔ **CHGAUT OBJ('/myroot/pkware/CStores/Root') USER(*PUBLIC) DTAAUT(*R) OBJAUT(*NONE)**
➔ **CHGAUT OBJ('/myroot/pkware/CStores/Root/pkwareRT.store') USER(*PUBLIC) DTAAUT(*R) OBJAUT(*NONE)**

Change ZIPADMN for all security:

➔ **CHGAUT OBJ('/myroot/pkware/CStores/Root/pkwareRT.store') USER(ZIPADMN) DTAAUT(*RWX) OBJAUT(*ALL)**
➔ **CHGAUT OBJ('/myroot/pkware/CStores/Root/pkwareRT.store') USER(ZIPADMN) DTAAUT(*RWX) OBJAUT(*ALL)**

If any other users or groups are defined for the path or file remove them by using the WRKLNK command and displaying authority.

```
                            Display Authority

Object . . . . . :   /myroot/pkware/CStores/Root/pkwareRT.store
Owner  . . . . . :   ZIPADMN
Primary group  . . . . . :   *NONE
Authorization list . . . . . :   *NONE


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

At this point you can use the PKSTORADM command to import the trusted roots and the intermediate certificates.

It is very important that you control the addition of trusted roots to your root store since they will define the certificate chains to be trusted.

Local Certificate Store

Smartcrypt for IBM i includes modules that utilize the OpenSSL Toolkit to access X.509 public and private key certificates. The access to the various certificate stores by this task is governed by various forms of the ENTPREC parameter, as well as by a suite of new configuration commands.

All local certificates are stored in a path of the Integrated File System (IFS), and the store paths are defined using the PKCFGSEC command. The other method of certificate access is through the LDAP server. The first step of the startup is to define the public and private stores and their security. The next part is to define a procedure to maintain the certificates.

Smartcrypt for IBM i provides access to both public and private key certificates through a set of local IFS paths which contain the certificate files. These paths are defined using the PKCFGSEC command.

This section assists with allocating the components necessary to support the local DB to administer the certificates within it.

Build Private and Public Store Paths

This section describes the process to build the public and private certificate paths in the IFS.

The public store is a path in the IFS that contains files with X.509 public certificate files known as PEM files.

The private store is a path in the IFS that contains a file with X.509 public certificate and private key packaged as a file.

Smartcrypt can work with certificates and keys in the following file formats:

Format	***Description***
PEM	Contains a single end-entity public-key certificate. It may be in Base-64 encoded (ASCII text with ASCII headers) or DER-encoded binary format. Common file extensions: `.pem`, `.cer`, `.key`, `.crt`
PKCS#12	Contains a single end-entity private-key certificate (and its public key). By definition, it is in binary format. Common file extensions: `.pfx`, `.p12`
PKCS#7 **Not Supported for public and private keys**	Contains one or more CA (and or Root) certificates Common file extension: `.p7b`

For example:

1. The Public and private store paths would have been defined with the PKSETPATHS program.
2. Define the public and private stores paths with the PKCFGSEC command using the

CSPUB and CSPRVT parameters.

Local Certificate Locator Database

Smartcrypt for IBM i comes with one physical file PKZCF1F and three logical files: PKZCF1LCN, PKZCF1LEM, and PKZCF1LPH. These files are part of the certificate locator database used for certificate processing. These files are activated by changing the security settings using the PKCFGSEC command. To use these files, see Chapter 8.

This locator database allows access to the public and private certificates using the common name, email address, or public hash key.

Here again, for security reasons, it is suggested that a specific user ID (such as ZIPADMN), user, or group be assigned to administrate the database. This can be accomplished by changing the security for the configuration file PKZCF1F and the logical files in the Smartcrypt library with the EDTOBJAUT command. First change *Public to only *Use for the files, and then set the administrator as the owner with security for changes. For example:

```
                          Edit Object Authority

Object . . . . . :   PKZCF1F         Owner  . . . . . :   ZIPADMN
  Library  . . . . :     PKW140XXS     Primary group  . . . :   *NONE
Object type  . . . . :   *FILE           ASP device . . . . . :   *SYSBAS

Type changes to current authorities, press Enter.

  Object secured by authorization list  . . . . . .   *NONE

                        Object    ----------Object-----------
User        Group       Authority  Opr  Mgt  Exist  Alter  Ref
*PUBLIC                 *USE        X
ZIPADMN                 *ALL        X    X     X      X     X
```

Initialize and Build Certificate Locator Database:

This section describes the process to build the certificate locator databases. The databases are distributed with product as empty files, but the DSS source is also distributed in case they ever need to be rebuilt. An easier way to rebuild the database is to clear the physical file member (CLRPFM).

1. Clear the file:

   ➔ **CLRPFM FILE(PKW14961S/PKZCF1F) MBR(*FIRST)**

   OR

   Build the databases with the CRTCERTDB CL program. CRTCERTDB will create the "iSeries Certificate Locator" physical file PKZCF1F and its associated logical files:

   - PKZCF1LCN "Certificate locator by CN (Common Name)"
   - PKZCF1LEM "Certificate locator by EM (Email Address)"
   - PKZCF1LFN "Certificate locator by FN (Friendly Name)"
   - PKZCF1LPH "Certificate locator by Public Hash"

The program will create the files in the target library parameter of the program such as Call CRTCERTDB 'PKW14961S'.

2. Define the paths and places for the certificates in the proper directories.
3. By using the PKCFGSEC command, update the certificate store settings for the certificates and set the CERTDB settings for the certificate database locator.
4. Run the PKBLDCDB as many times as necessary to process your certificates in the IFS stores.
5. Run the PKQRYCDB to view the certificates in the locator database.

The following stores are maintained by the utility PKBLDCDB:

Store	*Description*
Public	A store for end-entity certificates used to identify encryption recipients or for authentication of digital signatures. Certificate files in this store contain only public keys; they do not contain private keys. ***Smartcrypt for IBM i*** represents these certificates as files held in the local certificate store path. Normally these IFS files might have a suffix of "cer" or "pem", but the suffix is not pertinent to Smartcrypt. Other system types may refer to this store as "Other People" or "Address Book"
Private	A store for end-entity certificate files with their respective private keys. Private keys are used to decrypt files or perform digital signing. ***Smartcrypt for IBM i*** represents these certificates as files held in the local certificate store path. Normally these IFS files might have a suffix of "P12" or "PFX", but the suffix is not pertinent to Smartcrypt. (Private keys in this store are encrypted using PKCS#8 format and PKCS#5 version 2.) Other system types may refer to this store as "Personal" or "MY Store"

LDAP Certificate Store

Smartcrypt for IBM i also provides access to public key certificates located in an external LDAP (Lightweight Directory Access Protocol) server via a TCP/IP network connection.

Each LDAP server to be used should be defined with the PKCFGSEC command.

Each certificate store is described in detail in the following sections.

Usage Notes:

It is recommended that, as you define your enterprise configuration, you build it in a CL source. This will provide the ability to reset the configuration quickly if it gets changed or destroyed. One way is to copy the DFTCFGSEC member of QCLSRC in your zip library to a secure source file for your custom changes. Then update this custom configuration CL as you build your system.

4 Getting Started with Certificate Store Management

Requires Smartcrypt

This chapter describes the processes that are required to set up the security environment for ***Smartcrypt for IBM i***. It includes a detailed walk-through with sample certificates and test stores.

The following security administration commands are referenced in this chapter: PKCFGSEC (see Chapter 6), PKSTORADM (see Chapter 10), PKBLDCDB (see Chapter 8), PKQRYCDB (see Chapter 9), and PKLDAPTST (see Chapter 7). For commonly asked security questions, see Chapter 5.

Configuration of Smartcrypt with Test Digital Certificates

Included with the Smartcrypt library is an archive containing two public certificates, two corresponding PKCS#12 private keys files, a test public certificate with private key for CRL testing, one CRL file to apply, three initialized stores, and several samples Inlist files. See the $readme.txt to see content description.

These certificates are provided to assist in initial certificate testing and are not needed to run Smartcrypt.

Note that the test certificates are *not* trusted certificates and were not issued by a trusted certificate authority. They should never be used for production or for securing any of your organization's sensitive data. They are provided for testing only.

The archives in the distribution library (assuming PKW14961S) would be the file named TSTCERTS with the member called TSTCERTS.

➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts')**

```
Archive:  PKW140XXS/TSTCERTS(TSTCERTS)   17424 bytes   22 files
  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
      785 Defl:F      324  59% 03-08-05 07:29 6ca9e42e $readme.txt
      654 Defl:F      489  25% 02-09-05 13:18 2d32433a crl1.crl
      786 Defl:F      658  16% 12-27-04 13:18 4458cfef pktestdb3.crt
     2106 Defl:F     2058   2% 12-27-04 13:18 1773a716 pktestdb3.p12
      786 Defl:F      660  16% 12-27-04 13:17 6bc95df5 pktestdb4.crt
     2106 Defl:F     2052   3% 12-27-04 13:18 ba20cadd pktestdb4.p12
     2106 Defl:F     2055   2% 03-07-05 12:41 b7f957bb pktestdb9.pfx
     3573 Defl:F     1882  47% 02-09-05 13:17 0c6cc373 pktica_ca.cer
     3565 Defl:F     1878  47% 02-09-05 13:15 60f503e4 pkwareCA.crt
       37 Defl:F       30  19% 02-17-05 11:48 274e6484 pkwareca.p7b
       37 Defl:F       30  19% 02-17-05 11:48 274e6484 pkwarecrl.p7b
     1192 Defl:F      691  42% 02-09-05 13:15 40f1ec95 pkwareroot.cer
       37 Defl:F       30  19% 02-17-05 11:48 274e6484 pkwarert.p7b
       49 Stored       49   0% 02-25-05 10:50 b5002c1b tstauth_db3.inlist
       51 Stored       51   0% 01-10-05 12:14 3a6ddda4 tstauth_mb3.inlist
       47 Stored       47   0% 02-25-05 10:50 18df8708 tstauth_mb4.inlist
       52 Stored       52   0% 02-25-05 10:50 7c830a06 tstpriv_db4.inlist
       45 Stored       45   0% 02-25-05 10:50 cddd548c tstpriv_mb3.inlist
       78 Defl:F       52  33% 02-25-05 10:50 e6816f7e tstpubl_1.inlist
       83 Defl:F       72  13% 02-25-05 10:51 f3e66e5b tstpubl_2.inlist
       55 Stored       55   0% 02-25-05 10:51 d110fa75 tstsign_db3.inlist
       51 Stored       51   0% 02-25-05 10:51 441dad6d tstsign_mb4.inlist
 --------          -------  ----                            -------
    18281            13311  27%                             22 files
SecureUNZIP  extracted     0 files                                    .
```

The following steps can be automated for testing using the included CL program TSTCERTS. The source is located in the QCLSRC file of the distribution library. Before using the TSTCERTS CL program, make any modification required for your environment and then compile the CL program. The TSTCERTS program requires two external parameters. The first parameter is the Smartcrypt library. The second parameter is the base path where the certificate store will reside (for example: '/myroot/pkware'). This base path must exist before running TSTCERTS.

Step 1: Define IFS Certificate Stores

First you will need to decide where the certificate stores will be located in the IFS and then create the following folders in that path. After creating the paths, take time to set all the proper authorities for each path. If an administrator will be assigned, you can set the group ID as the owner. It is recommend that *PUBLIC should only have read authority to the files once you enter production. Assuming that you will be creating the stores in the directory called '/myroot/pkware', create the following directories in that base path:

- **MD DIR('/myroot/pkware/CStore') Be sure to define the folder security**
- **CALL PKSETPATHS ('PKW14961S' '/myroot/pkware/CStore')**
  Program will create the following folders:

'/myroot/pkware/CStore/Public'	**Public Store**
'/myroot/pkware/CStore/Private'	**Private Store**
'/myroot/pkware/CStore/CA'	**Certificate Authority Store**
'/myroot/pkware/CStore/Root'	**Trusted Authority Root Store**
'/myroot/pkware/CStore/CRL'	**Certificate Revocation List**
'/myroot/pkware/CStore/CTemp'	**Admin Temp Work Area**

- **MD DIR('/myroot/pkware/CStore/Testzips')** — **For initial archives for testing and inputs to admin**
- **MD DIR('/myroot/pkware/CStore/InList')** — **Used for testing Inlist samples**

The contents can be deleted after the test is completed, but the enterprise configuration should be reviewed for production.

Note: if you need to start over and want to clear the certificate locator database for testing, you can clear the database with:

➔ **CLRPFM FILE(PKW14961S/PKZCF1F) MBR(*FIRST).**

Caution should be taken to make sure you are clearing a test-only database.

Step 2: Extract Test Certificates, Inlist, Inputs, and Stores

Extract public test certificates:

➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts')**
FILES('pktestdb3.crt' 'pktestdb4.crt')
TYPE(*EXTRACT) EXDIR('/myroot/pkware/CStore/Public')
DROPPATH(*ALL) TYPARCHFL(*DB) TYPFL2ZP(*IFS) CVTTYPE(*NONE)

Extract private test certificates:

➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts')**
FILES('pktestdb3.p12' 'pktestdb4.p12' 'pktestdb9.pfx')
TYPE(*EXTRACT) EXDIR('/myroot/pkware/CStore/Private')
DROPPATH(*ALL) TYPARCHFL(*DB) TYPFL2ZP(*IFS) CVTTYPE(*NONE)

Extract initialized empty CA certificate intermediate store:

➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts') FILES('pkwareCA.store')**
TYPE(*EXTRACT) EXDIR('/myroot/pkware/CStore/CA')
DROPPATH(*ALL) TYPARCHFL(*DB) TYPFL2ZP(*IFS) CVTTYPE(*NONE)

Extract initialized empty trusted root store:

➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts') FILES('pkwareRT.store')**
TYPE(*EXTRACT) EXDIR('/myroot/pkware/CStore/Root')
DROPPATH(*ALL) TYPARCHFL(*DB) TYPFL2ZP(*IFS) CVTTYPE(*NONE)

Extract initialized empty revocation list store:

➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts') FILES('pkwareCRL.store')**
TYPE(*EXTRACT) EXDIR('/myroot/pkware/CStore/CRL')
DROPPATH(*ALL) TYPARCHFL(*DB) TYPFL2ZP(*IFS) CVTTYPE(*NONE)

Extract sample test Inlist files:

➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts')**
FILES('*inlist')
TYPE(*EXTRACT) EXDIR('/myroot/pkware/CStore/InList')
DROPPATH(*ALL) TYPARCHFL(*DB) TYPFL2ZP(*IFS) CVTTYPE(*NONE)

Restore test inputted certs and CRL to Testzips area

➔ **PKUNZIP ARCHIVE('PKW14961S/tstcerts/tstcerts')**
FILES('pkwareCA.cer' 'pkwareRoot.cer'
'pktica_ca.crt' 'crl1.crl')
TYPE(*EXTRACT) EXDIR('/myroot/pkware/CStore/Testzips')
DROPPATH(*ALL) TYPARCHFL(*DB) TYPFL2ZP(*IFS) CVTTYPE(*NONE)

Step 3: Setup Security Configuration File with PKCFGSEC Command

First, verify what the current settings in the security configuration with:

➔ **PKCFGSEC TYPE(*VIEW)**

Next, update the certificate stores and activate the certificate locator database.

1. Activate signing and authentication with file name encryption optional.

   ➔ **PKCFGSEC TYPE(*Update)**
   SECTYPE(*YES *YES *SAME *SAME *YES *YES *YES *YES *OPT)

2. Define all of the store paths and files with CSPUB, CSPRVT, CSCA, CSROOT, CSCRL.
3. Define the certificate database settings with CERTDB. Make certain that the correct library is specified or errors will occur later.
4. Define certificate policies for signing (SIGNPOL), encryption (ENCRYPOL), and authentication (AUTHPOL).
5. Define the static CRL settings with REVKEPOL.
6. We could define a contingency key at this time with ENTPREC

   ➔ **PKCFGSEC TYPE(*Update)**
   SECTYPE(*YES *YES *SAME *SAME *YES *YES *YES *YES *OPT)
   CERTDB(*YES *LIB 0 PKW14961S *EMAIL)
   CSPUB(*PATH 0 '/myroot/pkware/CStore/Public')
   CSPRVT(*PATH 0 '/myroot/pkware/CStore/Private')
   CSCA(*FILE 0 '/myroot/pkware/CStore/CA/pkwareCA.store')
   CSROOT(*FILE 0 '/myroot/pkware/CStore/ROOT/pkwareRT.store')
   CSCRL(*FILE 0 '/myroot/pkware/CStore/CRL/pkwareCRL.store')
   CWRKPATH('/myroot/pkware/CStore/CTemp')
   REVKEPOL(*STATICCRL *NONE)
   SIGNPOL(*SHA1 *VALIDATE *NO (*ALL))
   AUTHPOL(*VALIDATE *NO (*ALL))
   ENCRYPOL(*VALIDATE *NO (*ALL))
   ENTPREC(*NONE) LDAPACTV(*NONE)

Step 4: Add Trusted Roots and Intermediate Certificates to Certificate Stores.

Using the PKSTORADM store administrator maintenance command, we should import all the trusted roots first, followed by all CA certificates.

But first we need to think about the security of our stores. Most installations will want to have a specific user, user ID, or group perform store maintenance and let the users of Smartcrypt have only read access to the stores. It is suggested that *PUBLIC be only allowed to read the stores. For example, if we have a user ID of ZIPADMIN that is the only user that has the authority to do maintenance, then the paths and stores might have a sample authority setting as follows:

```
Object . . . . . . . . . . . . . :   /myroot/pkware/CStore/Root/pkwareRT.store
                   Data     --Object Authorities--
Opt  User         Authority  Exist  Mgt  Alter  Ref

     *PUBLIC      *RX          X
     ZIPADMIN     *RWX         X     X     X     X
```

First you might want to verify all the stores to make sure they are OK and empty.

➔ **PKSTORADM RUNTYPE(*VERIFY) STYPE(*ALL)**

```
PKSTORADM Smartcrypt CA Store Administration starting------2016/03/10 12:10:19

ZPCA000I SUCCESS: CA Store '/myroot/pkware/CStore/CA/pkwareCA.store' verified successfully.
0 certificates found.

ZPCA000I SUCCESS: Root Store '/myroot/pkware/CStore/ROOT/pkwareRT.store'verified
successfully.  0 certificates found.

ZPCA000I SUCCESS: CRL store '/myroot/pkware/CStore/CRL/pkwareCRL.store' successfully
verified.  0 revocation lists found.

PKSTORADM Store Admin ending------0
PKSTORADM  Completed Successfully
```

Now add the PKWARE test trusted root certificate to the root store:

➔ **PKSTORADM RUNTYPE(*IMPORT) FTYPE(*CER) STYPE(*ROOT) FNAME('/myroot/pkware/CStore/Testzips/pkwareRoot.cer')**

```
PKSTORADM Smartcrypt CA Store Administration starting------2016/03/10 12:10:19
ZPCA000I SUCCESS: Added certificate to store '/myroot/pkware/CStore/Root/
pkwareRT.store'.  DSN=/myroot/pkware/CStore/Testzips/pkwareRoot.cer;CER=1;F
N=PKTESTDB Root;TP=A91CD312EFFEF51C8ABFEFD92C23FF67FBDBEBE3;SN=00;E=PKTESTDBR
oot@nowhere.com;ROOT

ZPCA000I SUCCESS: Saved certificate store '/myroot/pkware/CStore/Root/pkw
areRT.store' to disk.

ZPCA000I Added 1 of a possible 1 certificates to the ROOT store.
ZPCA000I 0 certificates in the ROOT store before the Add command.
ZPCA000I 1 certificates in the ROOT store after the Add command.

PKSTORADM Store Admin ending------0
```

Next add the PKWARE test intermediate certificate authority certificate to the CA store:

➔ **PKSTORADM RUNTYPE(*IMPORT) FTYPE(*CER) STYPE(*CA) FNAME('/myroot/pkware/CStore/Testzips/pkwareCA.cer')**

```
PKSTORADM Smartcrypt CA Store Administration starting------2016/03/10 12:12:36
ZPCA000I SUCCESS: Added certificate to store '/myroot/pkware/CStore/CA/pk
wareCA.store'.  DSN=/myroot/pkware/CStore/Testzips/pkwareCA.cer;CER=1;FN=PK
WARE Test Intermediate Cert;TP=2B3C27C30067690EFB77A8A84DAB1D39A1368906;SN=01
;E=PKTESTDBIC@nowhere.com;CA

ZPCA000I SUCCESS: Saved certificate store '/myroot/pkware/CStore/CA/pkwar
eCA.store' to disk.

ZPCA000I Added 1 of a possible 1 certificates to the CA store.
ZPCA000I 0 certificates in the CA store before the Add command.
ZPCA000I 1 certificates in the CA store after the Add command.

PKSTORADM Store Admin ending------0
```

Step 4.a: Signing Algorithm Certificate Considerations.

The installation of a certificate that was generated with an internal signing hash algorithm of SHA-256 will create an operational incompatibility with earlier versions of Smartcrypt (See "Support for Digital Certificates with the SHA-256 RSA Signature Algorithm"). This can occur either when an i5/OS Security Server enforces the generation of certificates with an internal signing hash of SHA-256, or when an external Certificate Authority provides such certificates and they are added for use with Smartcrypt.

When an environment is known to only use software releases that support the SHA-256 RSA signing algorithm for digital certificates, they can CALL PGM(PKCSCHKR) PARM(CNTL01) to mark the Smartcrypt Key Store index for SHA-256 processing. This will eliminate administrative prompts from occurring when using the certificate add procedure.

Step 5: Build the Certificate Locator Database with PKBLDCDB

First build the public keys. In this case, we can use the *DIRECTORY option since all the public certificates are in one folder. You can use your user ID for the new owner or the ZipAdmin. If you do not have proper authority, then you will receive an API error when the CHGAUT command is issued. The database is ok, but the authority of the file may not be what you intended.

➔ **PKBLDCDB MAINT(*UPDATE) MTYPE(*DIRECTORY) CTYPE(*PUBLIC) FNAME('/myroot/pkware/CStore/Public') DUPOPT(*REPLACE) LOGLVL(*LOG) OWNER(*NONE) PUBAUTH(*R (*OBJREF *OBJEXIST))**

```
PKBLDCDB Update Smartcrypt Cert DataBase starting------2016/03/10 10:47:15
Sample Authorty:<CHGAUT OBJ('FileObj') USER(*PUBLIC) DTAAUT(*R) OBJAUT(*OBJRE
F *OBJEXIST  )>
PKBLDCDB Running Update Mode--Replace Duplicates---
PKBLDCDB Adding  </myroot/pkware/CStore/Public/pktestdb3.crt>
PKBLDCDB Adding  </myroot/pkware/CStore/Public/pktestdb4.crt>
PKBLDCDB Run Totals:
   Total Records Added     =2
   Total Records Updated   =0
   Total Duplicates Found      =0
   Total Records In Error  =0
   Total Records Processed =2
PKBLDCDB Scan ending------
```

Usually when updating private certificates you will use the *FILE option since each private key will have a different passphrase. The best practice is to have the individual to whom the certificate was issued run the PKBLDCDB command since, for security reasons, he or she should be the only one with access to the passphrase used to export the certificate file.

➔ **PKBLDCDB MAINT(*UPDATE) MTYPE(*FILE) CTYPE(*PRIVATE)**
FNAME('/myroot/pkware/CStore/Private/pktestdb4.p12')
DUPOPT(*REPLACE) PASSWORD('PKWARE')
LOGLVL(*LOG) OWNER(*NONE) PUBAUTH(*R (*OBJREF *OBJEXIST))

```
PKBLDCDB Update Smartcrypt Cert DataBase starting------2016/03/10 10:53:17
Sample Authority:<CHGAUT OBJ('FileObj') USER(*PUBLIC) DTAAUT(*R) OBJAUT(*OBJRE
F *OBJEXIST  )>
PKBLDCDB Running Update Mode--Replace Duplicates---
PKSCANCRT  Private Cert will be processed  (6)
PKBLDCDB Adding  </myroot/pkware/CStore/Private/pktestdb1.p12>
PKBLDCDB Run Totals:
   Total Records Added     =1
   Total Records Updated   =0
   Total Duplicates Found      =0
   Total Records In Error  =0
   Total Records Processed =1
PKBLDCDB Scan ending------
```

➔ **PKBLDCDB MAINT(*UPDATE) MTYPE(*FILE) CTYPE(*PRIVATE)**
FNAME('/myroot/pkware/CStore/Private/pktestdb3.p12')
DUPOPT(*REPLACE) PASSWORD('PKWARE')
LOGLVL(*LOG) OWNER(*NONE) PUBAUTH(*R (*OBJREF *OBJEXIST))

➔ **PKBLDCDB MAINT(*UPDATE) MTYPE(*FILE) CTYPE(*PRIVATE)**
FNAME('/myroot/pkware/CStore/Private/pktestdb9.pfx')
DUPOPT(*REPLACE) PASSWORD('PKWARE')
LOGLVL(*LOG) OWNER(*NONE) PUBAUTH(*R (*OBJREF *OBJEXIST))

Step 6: 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('PKW14961S/$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 PKW14961S/$CONTACT($CONTACT) in TEXT mode
Add  PKW14061.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('PKW14961S/$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 PKW14961S/$CONTACT($CONTACT) in TEXT mode
Updating:PKW14061.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
```

Step 7: View Details of Archive

By performing a PKUNZIP with TYPE(*VIEW) VIEWOPT(*ALL), you will see the recipients, and the number of recipients, that can be found in the certificate database. It will also list each recipient and show each common name and email address.

```
Archive:  /myroot/pkware/CStore/Testzips/TestC01.zip   2259 bytes   1 file
Archive Comment:"PKZIP for IBM i by PKWARE"
Filename: PKW14061.S/$CONTACT/$CONTACT
Detected File type:     Text   Encrypt=Strong Encrypted
Created by:             PKZIP(R) for IBM i  10.0
Minimum to Extract:     PKZIP 5.1  Or Greater
Compression method:     Deflated   [Fast]
Date and Time           2010 Jul 23 10:10:56
Compressed size:        1702 bytes
Uncompressed size:      5451 bytes
32-bit CRC value (hex): f091572d
Extended attributes:    yes, [Length = 218]
Strong Encryption AES 256 Key
Algorithm Key 256,  Security type Certificate Key
Number Certificate Recipients 2
Recipient List:
  CN=PKWARE Test4,  EM=PKTESTDB4@nowhere.com
  CN=PKWARE Test3,  EM=PKTESTDB3@nowhere.com
Record Length: 80
Lib Text Desc: Smartcrypt(R) for IBM i Lib-V16.0
File Text Desc: Smartcrypt(R) for IBM i Contact Information
Mbr Text Desc: Smartcrypt(R) for IBM i Contact Information
Source or Data: PF-DTA
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 1 file, 5451 bytes uncompressed, 1702 bytes compressed:   69%
SecureUnZip  extracted     0 files
SecureUnZip   Completed Successfully    1
```

Step 8: Decrypting File with Private Key Certificates

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')))
```

OR

```
➔ENTPREC((*MBRSET 'pktestdb3.p12' ('PKWARE')))
  ENTPREC((*FILE '/myroot/pkware/CStore/Public/pktestdb4.p12'
   ('PKWARE')))
  ENTPREC((*DB 'EM=PKTESTDB4@nowhere.com' ('PKWARE')))
  ENTPREC((*INLIST
   '/myroot/pkware/CStore/InList/tstpriv_mb3.inlist'))
```

Step 9: Using Input List File for ENTPREC Certificate List

The encryption recipient parameter can read a file with a recipient list. This is activated by using the *INLIST followed by the file to use for the recipient list.

path/filename

Enter the file path and name of the file to read. The layout depends on which file system you want to read the file from.

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

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

Usage Notes:

- For an 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.
- The path/filename depends on the settings of the TYPLISTFL (*DB or *IFS) parameter.
- For more information on using list files, see the *User's Guide*.

Each entry on a list file should begin with "{RECIPIENT=" and end with the character "}". Each entry is separated by the ";" (semi-colon). The information for each recipient is the same as described in the ENTPREC parameter, except *INLIST is invalid for lookup type. If an entry is not required such as passphrase just enter the separator.

Format: {RECIPIENT=Lookup Type;Recipient;Passphrase;Required}

```
*...+....1....+....2....+....3....+....4....+....5....+....6..
{RECIPIENT=db;EM=robb.roy@pkware.com;;RQD}
{RECIPIENT=file;/yourpath/PKWARE/Cstores/Public/suesmith04.cer;;OPT}
{RECIPIENT=LDAP;EM=kevin.Goodguy@pkware.com;;OPT}
{RECIPIENT=LDAP;EM=troy.theman@pkware.com;;OPT}
                          ****** END OF DATA ******
```

The ENTPREC parameter would look like the following to process the ENTPREC Inlist:

➔ **ENTPREC((*INLIST 'ATEST/INLIST(techsup1)' *N))**

Test Inlist tstpriv_mb3.inlist:

```
....+....1....+....2....+....3....+....4....+....5..
 ************Beginning of data*************
{AUTHCHK=ARCHIVE;MBRSET;pktestdb3.pfx;PKWARE;RQD}
 ************End of Data*******************
```

Test inlist tstpriv_db4.inlist:

```
....+....1....+....2....+....3....+....4....+....5.
 ************Beginning of data*************
{RECIPIENT=DB;EM=PKTESTDB4@nowhere.com;PKWARE;RQD}
 ************End of Data*******************
```

An example to decrypt a file that was encrypted with pktestdb3 public certificate.

➔ **PKUNZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC02.zip')**
TYPE(*TEST) TYPARCHFL(*IFS) TYPLISTFL(*IFS)
ENTPREC((*inlist '/myroot/pkware/CStore/InList/tstpriv_mb3.inlist'))

```
Digital Certificate Request List:Encryption Recipients
Rqrd    *MbrSet                    - pktestdb3.p12
Encryption Recipients List----------1 processed:
UNZIP Archive: /myroot/pkware/CStore/Testzips/TestC01.zip
Searching Archive /myroot/pkware/CStore/Testzips/TestC01.zip  for files to extract
Testing: WSS820S/$CONTACT/$CONTACT
WSS820S/$CONTACT/$CONTACT tested OK
No errors detected in compressed data of /myroot/pkware/CStore/Testzips/TestC01.zip.
SecureUNZIP   Completed Successfully
```

Step 10: Sign Files and Archive with Private Keys

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('PKW14961S/$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
```

Step 11: Authenticate Signed Files and Archive

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% 01-11-05 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
```

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

```
Archive:  /myroot/pkware/CStore/Testzips/TestC03.zip   6993 bytes   1 file

Filename: PKW140XX.S/$CONTACT/$CONTACT
Detected File type:     Text   Encrypt=Strong Encrypted
Created by:             PKZIP(R) for IBM i  10.0
Zip Spec to Extract:    6.1  Or Greater
Compression method:     Deflated   [Fast]
Date and Time           2005 Jan 11 16:05:36
Compressed size:        1702 bytes
Uncompressed size:      5451 bytes
32-bit CRC value (hex): f091572d
Extended attributes:    yes, [Length = 4598]
Strong Encryption AES 256 Key.
Algorithm Key 256,  Security type Certificate Key
Number Certificate Recipients 2
Recipient List:
```

```
  CN=PKWARE Test3,  EM=PKTESTDB3@nowhere.com
  CN=PKWARE Test4,  EM=PKTESTDB4@nowhere.com
Record Length: 80
Lib Text Desc: Smartcrypt(R) for IBM i Lib-V16.0
File Text Desc: Smartcrypt(R) for IBM i Contact Information
Mbr Text Desc: Smartcrypt(R) for IBM i Contact Information
Source or Data: PF-DTA
A subfield with ID 0x0014  (Certificate Store)  and 3369 data bytes
A subfield with ID 0x0016  (Archive Signature)  and 245 data bytes
File Signature: CN=PKWARE Test4; EM=PKTESTDB4@nowhere.com; S/N=04
File Signature: CN=PKWARE Test3; EM=PKTESTDB3@nowhere.com; S/N=03
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 1 file, 5451 bytes uncompressed, 1702 bytes compressed:   69%
Archive has been Digitally Signed.
Archive was signed by "PKWARE Test4" and verified
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

Do a VIEWOPT(*SECURE) and you will notice that the internal archive store is displayed (see certificate listing). This store contains all public certificates of the signatories including the full chain of each signatory.

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

```
Archive:  /myroot/pkware/CStore/Testzips/TestC03.zip   6993 bytes   1 file

  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
     5451 Defl:F     1702  69% 01-11-05 16:05 AES256 !PKW140XX.S/$CONTACT/$CONTACT
 --------         -------  ----                          -------
    5451             1702  69%                           1 file
Archive:  /myroot/pkware/CStore/Testzips/TestC03.zip   6993 bytes   1 file

Certificate Listing:
 #1:CN=PKWARE Test3; EM=PKTESTDB3@nowhere.com;
 #2:CN=PKWARE Test4; EM=PKTESTDB4@nowhere.com;
 #3:CN=PKTESTDB Root; EM=PKTESTDBRoot@nowhere.com;
 #4:CN=PKWARE Test Intermediate Cert; EM=PKTESTDBIC@nowhere.com;
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
```

Step 12: Sign Files and Archive with PK Test 9 Certificate

Try signing an archive with the PKWARE test 9 certificate, which does not have an intermediate certificate in the CA store. The attempt will fail because of the trust settings.

➔ **PKZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC04.zip')**
FILES('PKW14961S/$CONTACT')
TYPARCHFL(*IFS) TYPFL2ZP(*DB)
SIGNERS((*archive *MBRSET 'pktestdb9.pfx' (PKWARE)))

```
Scanning files in *DB for match  ...
Required Signer or Authenticator failed selection process
Digital Certificate Request List:Archive Signer
Rqrd    *MbrSet Not Usable         - pktestdb9.pfx
Required Recipient<pktestdb9.pfx> Failed
Archive Signer List----------0 processed:
CN=PKWARE Test9 EMail=PKTESTDB9@nowhere.com
--Certificate Invalid:Certificate is Not Trusted;
Smartcrypt Compressed 0 files in Archive /myroot/pkware/CStore/Testzips/TestC04.zip
Smartcrypt Completed with Errors
Press ENTER to end terminal session.
```

Step 13: Add CA to Make the Trust Valid and Then Sign Archive.

If we now add the intermediate certificate that “PKWARE test 9” certificate uses, the PKZIP command will sign the archive.

Add new CA certificate to the CA store:

➔ **PKSTORADM RUNTYPE(*IMPORT) FTYPE(*CER) STYPE(*CA)**
FNAME('/myroot/pkware/CStore/Testzips/pktica_ca.crt')

```
ZPCA000I SUCCESS: Added certificate to store '/myroot/pkware/CStore/CA/pk
wareCA.store'.  DSN=/myroot/pkware/CStore/Testzips/pktica_ca.crt;CER=1;FN=P
KWARE Test Intermediate Cert A;TP=C405A5BC36942149D5C212CB6C213C4F1FE893E6;SN
=02;E=PKTESTDBICA@nowhere.com;CA

ZPCA000I SUCCESS: Saved certificate store '/myroot/pkware/CStore/CA/pkwar
eCA.store' to disk.

ZPCA000I Added 1 of a possible 1 certificates to the CA store.

ZPCA000I 1 certificates in the CA store before the Add command.

ZPCA000I 2 certificates in the CA store after the Add command.
```

```
PKSTORADM Store Admin ending------0
Press ENTER to end terminal session.
```

Run PKZIP to sign the archive with PK test certificate 9.

➔ **PKZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC04.zip')**
FILES('PKW14961S/$CONTACT')
TYPARCHFL(*IFS) TYPFL2ZP(*DB)
SIGNERS((*archive *MBRSET 'pktestdb9.pfx' (PKWARE)))

```
Scanning files in *DB for match  ...
Digital Certificate Request List:Archive Signer
Rqrd    *MbrSet                  - pktestdb9.pfx
Archive Signer List----------1 processed:
CN=PKWARE Test9 EMail=PKTESTDB9@nowhere.com
Found  1 matching files
Compressing PKW140XXS/$CONTACT($CONTACT) in TEXT mode
Add  PKW140XX.S/$CONTACT/$CONTACT  --  Deflating (80%)
Smartcrypt Compressed 1 files in Archive /myroot/pkware/CStore/Testzips/Te
stC04.zip
Smartcrypt Completed Successfully
Press ENTER to end terminal session.
```

Run PKUNZIP to test the authentication for a specific authenticator.

➔ **PKUNZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC04.zip')**
TYPARCHFL(*IFS) TYPFL2ZP(*DB)
AUTHCHK((*ALL *MBRSET 'pktestdb9.pfx' (PKWARE)))

```
Digital Certificate Request List:File Signers
Rqrd    *MbrSet                  - pktestdb9.pfx
File Signers List----------1 processed:
Digital Certificate Request List:Archive Signer
Rqrd    *MbrSet                  - pktestdb9.pfx
Archive Signer List----------1 processed:
Archive:  /myroot/pkware/CStore/Testzips/TestC04.zip   4769 bytes   1 file

  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
     5451 Defl:F     1069  80% 03-08-05 07:38 f091572d PKW140XX.S/$CONTACT/$C
ONTACT
 --------         -------  ----                            -------
    5451             1069  80%                             1 file
Archive has been Digitally Signed.
Archive was signed by "PKWARE Test9" and verified
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

Step 14: Revoke PK Test 9 Certificate and Confirm Revocation

Now apply the certificate revocation list to revoke the PKWARE test 9 certificate.

➔ **PKSTORADM RUNTYPE(*IMPORT) FTYPE(*CRL) STYPE(*CRL)**
FNAME('/myroot/pkware/CStore/Testzips/crl1.crl')

```
PKSTORADM Smartcrypt CA Store Administration starting------2005/03/08 15:20:38
ZPCA000I SUCCESS: Added certificate '/myroot/pkware/CStore/Testzips/crl1.
crl' to store '/myroot/pkware/CStore/CRL/pkwareCRL.store'.

ZPCA000I SUCCESS: Saved certificate store '/myroot/pkware/CStore/CRL/pkwa
reCRL.store' to disk.

ZPCA000I Added 1 out of 1 certificates to the CRL store.
ZPCA000I 0 entries in the CRL store before the Add command.
ZPCA000I 1 entries in the CRL store after the Add command.

PKSTORADM Store Admin ending------0
Press ENTER to end terminal session.
```

Run PKZIP to test the authentication and signing. The attempt will fail since the signing certificate was revoked.

➔ **PKZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC04.zip')**
FILES('PKW14961S/$CONTACT')
TYPARCHFL(*IFS) TYPFL2ZP(*DB)
SIGNERS((*archive *MBRSET 'pktestdb9.pfx' (PKWARE)))

```
Scanning files in *DB for match  ...
Required Signer or Authenticator failed selection process
Digital Certificate Request List:Archive Signer
Rqrd    *MbrSet Not Usable         - pktestdb9.pfx
Required Recipient<pktestdb9.pfx> Failed
Archive Signer List-----------0 processed:
CN=PKWARE Test9 EMail=PKTESTDB9@nowhere.com
--Certificate Invalid:Certificate has been Revoked;
Input Archive was found to be Digitally Signed.
Archive Authentication check unsuccessful
Archive was Signed by "PKWARE Test9"  but  NOT verified.
--Contents have not been altered
--Certificate Invalid:Certificate has been Revoked;
Run being terminated for Authentication failure
Smartcrypt Compressed 0 files in Archive /myroot/pkware/CStore/Testzips/Te
stC04.zip
Smartcrypt Completed with Errors
```

Run PKUNZIP to test the authentication. The test will fail since the signing certificate was revoked. If we did not have the AUTHPOL (*SYSTEM *ALL), you would only see that archive was signed. The AUTHPOL is what activates the authentication

➔ **PKUNZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC04.zip')**
TYPARCHFL(*IFS) TYPFL2ZP(*DB)
AUTHCHK((*ALL *MBRSET 'pktestdb9.pfx' (PKWARE)))
AUTHPOL(*SYSTEM *ALL (*NOTREVOKED))

```
Digital Certificate Request List:File Signers
Rqrd    *MbrSet                    - pktestdb9.pfx
File Signers List-----------1 processed:
Digital Certificate Request List:Archive Signer
Rqrd    *MbrSet                    - pktestdb9.pfx
Archive Signer List-----------1 processed:
Archive:  /myroot/pkware/CStore/Testzips/TestC04.zip   4769 bytes   1 file
```

```
  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
     5451 Defl:F     1069  80% 03-08-05 07:38 f091572d PKW140XX.S/$CONTACT/$C
ONTACT
 --------         -------  ----                            -------
    5451            1069  80%                             1 file
Archive has been Digitally Signed.
Archive Authentication check unsuccessful
Archive was Signed by "PKWARE Test9"  but  NOT verified.
--Contents have not been altered
--Certificate Invalid:Certificate has been Revoked;
SecureUNZIP  extracted     0 files
SecureUNZIP Completed with Errors
```

Run PKUNZIP to test the authentication with a revoked signer but using AUTHPOL to ignore the revoked status of the signer.

➔ **PKUNZIP ARCHIVE('/myroot/pkware/CStore/Testzips/TestC04.zip') TYPARCHFL(*IFS) TYPFL2ZP(*DB) AUTHPOL(*SYSTEM *ALL (*NOTREVOKED))**

```
Archive:  /myroot/pkware/CStore/Testzips/TestC04.zip   4769 bytes   1 file
  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
     5451 Defl:F     1069  80% 03-08-05 07:38 f091572d PKW140XX.S/$CONTACT/$C
ONTACT
 --------         -------  ----                            -------
    5451            1069  80%                             1 file
Archive has been Digitally Signed.
Archive was signed by "PKWARE Test9;" and verified
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

Directory Certificate Store Configuration - LDAP

This section assists with defining the network connectivity to access certificate stores in an LDAP-compliant directory. Certificate stores are often located on an LDAP server. Please note that network connections must be defined prior to using LDAP services to locate public key digital certificates for recipient processing.

Command settings are kept in a LDAP ***Smartcrypt for IBM i*** enterprise security configuration file.

To assist in entering the correct settings that are defined in the configuration file using the PKCFGSEC command, there is a LDAP test command (PKLDAPTST). This command verifies that the settings are correct and shows the type of response to expect. With PKLDAPTST, the LDAP connection parameters can be coded manually. However, the PKCFGSEC command has the ability to run the LDAP test to assist in properly formatting the LDAP parameters and to test connectivity to the desired LDAP server.

Testing the LDAP Connection

For more information, see the PKLDAPTST command in Chapter 7.

The presence of a server can be established by using the PING and PKLDAPTST command:

```
                    Smartcrypt LDAP Test       (PKLDAPTST)

Type choices, press Enter.

Action Type  . . . . . . . . . .   *SUMMARY      *SUMMARY, *DETAIL
Server Name or IP  . . . . . . . > '192.168.132.25'
Port Number  . . . . . . . . . .   389           Character value
Access User  . . . . . . . . . . > 'PKWAREADDEV\test'
Access Passphrase. . . . . . . .
Start Node . . . . . . . . . . . > 'cn=users,dc=pkwareaddev,dc=com'

Search Filter  . . . . . . . . .   '(cn=*)(userCertificate=*)'
Max Item . . . . . . . . . . . . > 100           Character value
```

Or

➔ **PKLDAPTST SNAME('192.168.132.25') USER('PKWAREADDEV\test') PASSWORD('xxxx') STRNODE('cn=users,dc=pkwareaddev,dc=com') MAXI(100)**

Sample results:

```
   PKLDAPTEST LDAP Test Starting       2012/07/22 06:23:34

 PKLDAPTESTT Parameters:Action<T>
          - Server<192.168.132.25> Port<389>
          - User<PKWAREADDEV\test> Passphrase<6>
          - Start Node<cn=users,dc=pkwareaddev,dc=com>
          - Search Filter<(&(cn=*)(userCertificate=*))>
          - Max Items to Search<100> ,100
 LDAP_intialTest - --LDAP init ..... elasp time 0.000000 seconds
 LDAP_intialTest - --SizeLimit=100  TimeLimit=0
 LDAP_intialTest - --LDAP bind ..... elasp time 0.000000 seconds
 LDAP_intialTest - --LDAP Search ..... elasp time 0.000000 seconds
 LDAP_intialTest - --LDAP Search ...... 21 entries found
 LDAP_intialTest - --LDAP Attributes ..... elasp time 0.000000 seconds
 LDAP_intialTest - Total Entries=21
 PKLDAPTESTT LDAP Testing Ending RC=0
```

To test an LDAP server defined to the system (with the PKCFGSEC command), enter

***LDAPn**

where n is the number (1 through 3) of the LDAP server. For example:

➔ **PKLDAPTST SNAME(*LDAP1)**

When creating a configuration for an LDAP server at a new network address, it is recommended that a PING test be performed first using the PING command.

OPTION - PING

The PING option will perform a "TSO PING" command to verify that the network address can be resolved and the associated IP address reached. Once completed, a BROWSE of the output will be automatically presented. Be aware that some network administrators may turn off PING response capabilities, so it is possible that the PING may time out even if the network name (e.g., www.pkware.com) can be resolved to an IP address.

➔ **PING '192.168.132.25'**

```
Verifying connection to host system 192.168.132.25.
PING reply 1 from 192.168.132.25 took 91 ms. 256 bytes. TTL 125.
PING reply 2 from 192.168.132.25 took 22 ms. 256 bytes. TTL 125.
PING reply 3 from 192.168.132.25 took 91 ms. 256 bytes. TTL 125.
PING reply 4 from 192.168.132.25 took 20 ms. 256 bytes. TTL 125.
PING reply 5 from 192.168.132.25 took 20 ms. 256 bytes. TTL 125.
Round-trip (in milliseconds) min/avg/max = 20/48/91.
Connection verification statistics: 5 of 5 successful (100 %).
```

Possible errors:

- The network address cannot be resolved by the domain name server (DNS).

TCP3202 Unknown host www.unknown-name.com

- Network services may be down along the routes to reach the IP address.

HOST unreachable

- The specified host may not be up, or is not accepting PING requests.

Timed out

File Name Encryption

How *Smartcrypt* Encrypts File Names

Smartcrypt for IBM i encrypts file names using your current settings for (strong) encryption method and algorithm. File names can be encrypted using strong passphrase encryption, 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.

Note: 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.

When Smartcrypt Encrypts File Names

With archives that do not already contain encrypted file names:

Smartcrypt for IBM i encrypts file names whenever you update an archive requesting FNE(*YES): including *Add, Freshen, Update, Delete* and *Copy.*

Smartcrypt for IBM i encrypts file names only when you provide encryption parameters such as PASSWORD and/or ENTPREC for recipients and set parameter FNE(*YES).

Encrypting File Names When You Update an Archive

If you turn on the setting to encrypt file names and then update an archive that already contains no encrypted files with unencrypted file names, ***Smartcrypt for IBM i*** encrypts all the file names in the archive.

If the archive contains any files whose contents are already encrypted, ***Smartcrypt for IBM i*** will bypass an attempt to add file name encryption.

If you update an archive that already contains files with encrypted file names, ***Smartcrypt for IBM i*** uses the current ADVCRYPT setting to determine whether file names should be encrypted in the output archive.

If FNE(*YES) is enabled when updating an archive already containing file name encryption, ***Smartcrypt for IBM i*** encrypts the new archive directory using the same passphrase or recipient list originally used to encrypt file names in the archive. Newly supplied values for PASSWORD or RECIPIENT are discarded. However, if a PASSWORD is used to access the input archive for the update process to begin, it must match the original passphrase used to create the encrypted archive directory.

Note: Once file names in an archive are encrypted, you cannot change the passphrase or recipient list used.

You cannot change the encryption method on files that are already in an archive that contains encrypted file names.

File name encryption is not compatible with GZIP and will not be used if requested.

Opening and Viewing an Archive that Has Encrypted File Names

An archive that contains encrypted file names requires ***SecureZIP for IBM i*** 8.2 or later (including Smartcrypt) to open it. To view the file name encrypted archive encryption detail and not the archive's content, use TYPE(*VIEW) VIEWOPT(*FNEALL).

The version number is not a version of the ***SecureZIP for IBM i*** program but rather a version of the ZIP file format. Version 8.2 of the program uses features of the 6.20 ZIP file format specification (ZipSpec) that are not available in earlier versions. Preceding versions of the program used earlier versions of the ZipSpec.

When opening an archive, ***Smartcrypt for IBM i*** automatically decrypts file names for anyone who is on the recipient list (or enters the correct passphrase if PASSWORD was specified when the archive was updated with ADVCRYPT).

If file names are encrypted using a passphrase (with or without a recipient list), ***Smartcrypt for IBM i*** requires the correct passphrase when anyone who is not on the recipient list tries to open the archive. If the correct passphrase is not entered, PKZIP does not open the archive.

Security Examples

Below are examples of how to invoke ***Smartcrypt for IBM i*** processing using PKZIP and PKUNZIP commands with sample output listings.

Smartcrypt using Recipients or Combo

When protection modes of “Recipient” or “Combo” are selected, recipients can be designated such that a passphrase is not required to extract the data.

If a passphrase is entered, the lines will be concatenated to create a single passphrase string of up to 260 characters and each line must begin and end with a non-blank.

Each recipient is represented by a public-key X.509 digital certificate. The public-key certificates can be stored and accessed in one or more of the following locations:

- Individual data sets in an IFS path
- The local certificate store database as described by DB profile
- One or more network LDAP servers as described by LDAP Profile

Recipient designations:

1. ENTPREC((*LDAP CN=Joe Smith))
2. ENTPREC((*file '/PKZIP/CERTSTOR/PRIVATE/MAS2010' 'abcdef' RQD))
3. ENTPREC((*db EM=somebody@somewhere.com))
4. A. ENTPREC((*MBRSET MAS2010 'abcdef' RQD))

   B. ENTPREC((*MBRSET MAS2010 '' RQD))

It is important to note the following:

- CN=Joe Smith may return more than one recipient digital certificate. The LDAP entry for Joe Smith may contain multiple certificates, such as one for each year of employment with the company, since certificates are frequently valid for only one year.
- A local IFS path has a certificate loaded into file MAS2010, which may represent a specific person's 2010 certificate. In this case, the RQD indicates that the certificate is required for processing to be performed. In addition, this certificate is a private key certificate, so the export passphrase is necessary for the public key portion to be extracted from it.
- *db;EM= (or CN= for common name) may be used to locate a public key certificate from within the local certificate store database. Private key certificates may also be stored in the database, in which case the correct private key passphrases are required to access them.
- *MBRSET demonstrates how to enter only the file name. The path will come from the store defined in the enterprise security configuration file. Since the 4A item has a passphrase entered, the private store will be used. The 4B item contains no passphrase therefore the public store will be used.

Smartcrypt Encryption Using a Recipients List

A. ➔ PKZIP ARCHIVE('/yourpath/aVXXTest/test010.zip')
FILES('/yourpath/aVXXTest/recp/Test cases.txt')
TYPARCHFL(*IFS) TYPFL2ZP(*IFS) STOREPATH(*NO)
ADVCRYPT(AES256) PASSWORD(pkware12) VPASSWORD(pkware12)
ENTPREC((*DB 'EM=Bill.Somebody@pkware.com' *N *RQD)
(*FILE '/yourpath/PKWARE/Cstores/pkware_testcerts/pktestdb3.crt')
(*MBRSET 'pktestdb3.crt'))

Notice there were three inputted recipient requests, and there were four recipients used for the archive.

First, the CN=PKWCADMIN was added because the enterprise security settings defined a contingency key that will be added when doing strong encryption.

Second, the EM=Bill.Somebody, found two certificates in the database locator.

Third, since the pktestdb3.crt file was found two times using two different access methods, we only used it once for the process.

Displayed output from example A.

```
Scanning files in *IFS for match  ...
Total Recipients processed 4
Archive Recipient List:
CN=PKWARE Test3 EMail=PKTESTDB3@nowhere.com
CN=PKWCADMIN EMail=none
CN=BILL SOMEBODY EMAIL=BILL.SOMEBODY@PKWARE.COM
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/test010.zip
```

Store

B. ➔ PKZIP ARCHIVE('/yourpath/aVXXTest/test011.zip')
FILES('/yourpath/aVXXTest/recp/Test cases.txt')
TYPARCHFL(*IFS) TYPFL2ZP(*IFS) STOREPATH(*NO)
ADVCRYPT(AES256) PASSWORD(pkware12) VPASSWORD(pkware12)
ENTPREC((*DB 'CN=Bill Somebody' *N *RQD)
(*FILE '/yourpath/PKWARE/Cstores/pkware_testcerts/pktestdb3.crt')
(*MBRSET 'pktestdb4.crt'))

In this example, notice the request is for CN= and not EM=. This resulted in only one certificate to process from this database request.

Displayed output from example B.

```
Scanning files in *IFS for match  ...
Total Recipients processed 4
Archive Recipient List:
CN=PKWARE Test2 EMail=PKTESTDB3@nowhere.com
CN=PKWARE Test1 EMail=PKTESTDB1@nowhere.com
CN=PKWCADMIN EMail=none
CN=Bill 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/test011.zip
Smartcrypt Completed Successfully
```

Store

C. ➔ PKZIP ARCHIVE('/yourpath/aVXXTest/test012.zip')
FILES('/yourpath/aVXXTest/recp/Test cases.txt')
TYPARCHFL(*IFS) TYPFL2ZP(*IFS) TYPLISTFL(*IFS)
STOREPATH(*NO) ADVCRYPT(AES256) PASSWORD(pkware12)
VPASSWORD(pkware12)
ENTPREC((*DB 'CN=Bill Somebody' *N *RQD)
(*INLIST '/inlist/tstpubl2.inlist'))

This example used the Inlist option to read in a list of recipients:

INLIST file '/inlist/tstpubl2.inlist':
*************Beginning of data***************
{RECIPIENT=DB;EM=PKTESTDB3@nowhere.com;;RQD}
{RECIPIENT=DB;CN=PKWARE Test4;;OPT}
*************End of Data********************

Displayed output from example C.

```
Scanning files in *IFS for match  ...
Total Recipients processed 4
Archive Recipient List:
CN=PKWCADMIN EMail=none
CN=Bill Somebody EMail=bill.Somebody@pkware.com
CN=PKWARE Test3 EMail=PKTESTDB3@nowhere.com
CN=PKWARE Test4 EMail=PKTESTDB4@nowhere.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/test012.zip
Smartcrypt Completed Successfully
```

Smartcrypt Encryption using LDAP search for Recipients

D. ➔ 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 D.

```
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
```

UNZIP Compressed File(s) from an Archive File Using Recipients

Since we will be extracting/testing files that were encrypted with our public keys, we will need the private key to decrypt the file. Below are several examples, with one being correct and the others showing possible errors with bad certificates or wrong passphrase for the private key.

Unzip Output using Recipients

To decrypt a file with recipients, a valid passphrase for the certificate is required. Below is a valid test of example A.

A. ➔ PKUNZIP ARCHIVE('/yourpath/aVXXTest/test010.zip')
TYPARCHFL(*IFS) TYPE(*TEST)
ENTPREC((*MBRSET 'pktestdb3.p12' 'PKWARE'))

Displayed output from test of example A.

```
Total Recipients processed 1
UNZIP Archive: /yourpath/aVXXTest/test010.zip
Searching Archive /yourpath/aVXXTest/test010.zip  for files to extract
Testing: Test cases.txt
Test cases.txt tested OK
No errors detected in compressed data of /yourpath/aVXXTest/test010.zip.
SecureUNZIP   Completed Successfully
```

B. ➔ PKUNZIP ARCHIVE('/yourpath/aVXXTest/test010.zip')
TYPARCHFL(*IFS) TYPE(*TEST)
ENTPREC((*MBRSET 'pktestdb3.cer'))

Displayed output from test of example B.

```
Required Recipient failed selection process
<pktestdb3.cer> Recipient Requires Passphrase for Private Key
Required Recipient<pktestdb3.cer> Failed
```

C. ➔ PKUNZIP ARCHIVE('/yourpath/aVXXTest/test010.zip')
TYPARCHFL(*IFS) TYPE(*TEST)
ENTPREC((*MBRSET 'pktestdb1.p12' 'PKWAREXX'))

Displayed output from test example **C** with a bad passphrase for the private key.

```
Required Recipient failed selection process
Inputted Recipient Listing
Rqrd    *MbrSet Cert Open Failure pktestdb3.p12
Required Recipient<pktestdb3.p12> Failed
```

View the Contents of an Archive File

When a file has been encrypted, one of the following indicators describing the strength of encryption is displayed before the file name.

A. ➔ PKUNZIP ARCHIVE('/yourpath/aVXXTest/test013.zip')
TYPARCHFL(*IFS) TYPE(*view)

Displayed output from example A.

```
Archive:  /yourpath/aVXXTest/test010.zip   2024 bytes   1 file
  Length  Method    Size  Ratio  Date   Time  CRC-32    Name
 -------- ------  ------- -----  ----   ----  ------    ----
     4234 Defl:F     1722  59% 03-23-04 14:04 11cc5542 !Test cases.txt
 --------         -------  ----                          -------
    4234             1722  59%                           1 file
```

Notice the "!" in front of the file in archive name. This indicates that this file is encrypted with strong encryption. A "+" would indicate that only 96-bit encryption was used.

View Detail Display

The View Detail option not only shows the encryption algorithm used to encrypt but also displays any certificate information that is available.

B. ➔ PKUNZIP ARCHIVE('/yourpath/aVXXTest/test010.zip')
TYPARCHFL(*IFS) TYPE(*view) VIEWOPT(*ALL)

Displayed output from example B.

```
Archive:  /yourpath/aVXXTest/test010.zip   2024 bytes   1 file
Filename: Test cases.txt
Detected File type:     Binary   Encrypt=Strong Encrypted
Created by:             PKZIP(R) for IBM i  10.0
Zip Spec to Extract:    6.1  Or Greater
Compression method:     Deflated   [Fast]
Date and Time           2010 Mar 23 14:04:06
Compressed size:        1722 bytes
Uncompressed size:      4234 bytes
32-bit CRC value (hex): 11cc5542
Unix file attributes (100777 octal):           -rwxrwxrwx
MS-DOS file attributes (00 hex):           none
Extended attributes:    yes, [Length = 138]
Strong Encryption AES 256 Key.
Algorithm Key 256,  Security type Passphrase and Certificate Key
Number Certificate Recipients 4
Recipient List:
  CN=PKWARE Test3,  EM=PKTESTDB3@nowhere.com
  CN=PKWCADMIN,  EM=(none)
  CN=Bill Somebody,  EM=bill.Somebody@pkware.com
  CN=William Somebody,  EM=bill.Somebody@pkware.com
IFS: Creation Time      Wed Mar 24 17:23:44 2009
IFS: Access Time        Thu Sep 16 19:41:11 2009
IFS: Modification Time  Tue Mar 23 19:04:06 2009
IFS: Code Page          1252
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 1 file, 4234 bytes uncompressed, 1722 bytes compressed:   59%
```

C ➔ PKUNZIP ARCHIVE('/yourpath/aVXXTest/test013.zip')
TYPARCHFL(*IFS) TYPE(*view) VIEWOPT(*ALL)

Displayed output from example C.

```
Archive:  /yourpath/aVXXTest/test013.zip   1660 bytes   1 file
Filename: Test cases.txt
Detected File type:     Binary   Encrypt=Strong Encrypted
Created by:             PKZIP(R) for IBM i  10.0
Zip Spec to Extract:    6.1  Or Greater
Compression method:     Deflated   [Fast]
Date and Time           2010 Mar 23 14:04:06
Compressed size:        1398 bytes
Uncompressed size:      4234 bytes
32-bit CRC value (hex): 11cc5542
Unix file attributes (100777 octal):           -rwxrwxrwx
MS-DOS file attributes (00 hex):           none
Extended attributes:    yes, [Length = 98]
Strong Encryption AES 256 Key.
Algorithm Key 256,  Security type Certificate Key
Number Certificate Recipients 2
Recipient List:
  CN=PKWCADMIN,  EM=(none)
  CN=William Somebody,  EM=bill.Somebody@pkware.com
```

```
IFS: Creation Time      Wed Mar 24 17:23:44 2009
IFS: Access Time        Thu Sep 16 20:10:44 2009
IFS: Modification Time  Tue Mar 23 19:04:06 2009
IFS: Code Page          1252
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 1 file, 4234 bytes uncompressed, 1398 bytes compressed:   67%
SecureUNZIP  extracted     0 file
```

5 Security Questions and Solutions

Requires Smartcrypt

This chapter contains answers to questions a system administrator is likely to have about integrating ***Smartcrypt for IBM i*** into the operating environment.

Which encryption settings should be chosen?

Various external factors such as legislative requirements or corporate policy may influence your decision to select an algorithm or mode of encryption. However, when operating within those requirements, the following PKZIP and Smartcrypt information may be of value.

- NIST has instructional information regarding passphrase versus certificate-based (PKI) encryption. In general, certificate-based encryption is considered more secure than passphrase-based encryption.
- With the exception of supporting the older 96-bit "Standard" PKZIP ADVCRYPT, newer algorithms are provided at a minimum of 128 bits.
- PKWARE provides interoperability between OS/390, zOS, OS400, iSeries, IBM i OS, UNIX and Windows for all algorithms provided with ADVCRYPT with its product set at release 8.0 and above. This includes more advanced algorithms with minimum key lengths of 128 bits.
- Older releases of PKZIP products support "Standard" 96-bit encryption for wider cross-platform compatibility when required.
- Passphrase-based AES encryption is supported by PKWARE products at release 5.5 or higher.
- AES passphrase-based encryption is 100% compatible across the PKWARE product line.
- The OpenSSL algorithms provided for the z/OS and IBM i products are high-performance algorithms. The 128-bit OpenSSL algorithms even out-perform the older 96-bit PKZIP "Standard" algorithm.
- Smartcrypt PKPGPU extracts and decrypts files that comply with the OpenPGP standard, RFC 4880. Smartcrypt PKPGPZ can also create OpenPGP-compliant files and sign files with OpenPGP certificates.
- OpenPGP archives support 3DES, AES and CAST5 encryption algorithms.

How does recipient-based encryption differ from passphrase?

Passphrase-based encryption depends on both the sender and receiver knowing, and providing 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; therefore 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.

What is a digital certificate store?

Recipient-based encryption requires that public and private key certificates be used by ***Smartcrypt for IBM i***. These are kept in file streams encoded according to the X.509 standard. A certificate store is the location of where various types of certificates are kept and accessed.

The primary stores used by ***Smartcrypt for IBM i*** include:

CSPUB: Certificate store for individual public-key X.509 certificates on the local system.

CSPRVT: Certificate store for individual private-key X.509 certificates on the local system.

CSCA: Certificate store For Certificate authority public-key X.509 certificates on the local system.

CSROOT: Certificate store for the trusted root public-key X.509 certificates on the local system.

CSCRL: Certificate revocation list store static CRL processing on the local system.

LDAP: Certificate store for individual public-key X.509 certificates accessible via a TCPIP network.

Can both recipient-based and passphrase encryption be used together?

Yes, when both ENTPREC for recipient and PASSWORD settings are used to encrypt a file, the master session key is derived from the passphrase and is also protected by using public key encryption. This means that a ZIP file may be decrypted either by a user who knows the passphrase, or by one whose private key is accessible.

How does encryption method (ADVCRYPT) pertain to recipient or passphrase encryption?

Public/private key encryption using OpenSSL is used to digitally envelope the master session key information. Once the master session key is determined, an independent file session key is derived (which is unique for each file) to encrypt the file data with a symmetric algorithm specified by ADVCRYPT. Several algorithms are supplied with ***Smartcrypt for IBM i***. Any algorithm may be specified for use with PASSWORD.

How is encryption activated?

Encryption is activated through the use of the PASSWORD and/or RECIPIENT parameters. Encryption will be attempted in accordance with other applicable settings (such as ADVCRYPT or FACILITY) if a value is present for either setting, whether through explicit options or default settings.

However, if ADVCRYPT(*NONE) is specified, then encryption is bypassed.

Note that certificate-based encryption for recipients are supported only by Smartcrypt, not by PKZIP. Certificate-based mode of encryption requires that one of the strong ADVCRYPTs (minimum 128-bit) be selected.

How many recipients can be specified?

The ZIP file format specification allows for a maximum recipient-list size of 3,275. This size can be restricted further by other file attributes associated with the data, and by run-time capacity limitations (such as virtual storage). (Note: Approximately 20 bytes is required for each recipient within the ZIP archive central directory record for each file. This area is limited to 64K in size).

How do we activate a contingency key or “contingency recipient”?

To meet corporate security policies, Smartcrypt provides a Contingency Key setting to include a “contingency key” certificate in a SECZIP job whenever strong encryption is activated. The setting causes the data to be encrypted for the contingency key recipient(s) in addition to other specified recipients or passphrase settings. This ensures that the organization can always decrypt its encrypted data by applying the “global” contingency key certificate.

The contingency key may be set directly using the ENTPREC parameter in the PKCFGSEC command.

```
Contingency Key(s) :
   LookUp Type  . . . . . .   *SAME         *NONE, *DB, *LDAP, *FILE...
   Recipient  . . . . . .
   Required . . . . . . . .   *SAME         *RQD, *OPT, *SAME
```

How do contingency keys affect activation?

When Smartcrypt is being used to encrypt data, either with ENTPREC or PASSWORD, a recipient specified as a contingency key is automatically included. However, contingency keys do not trigger encryption to take place.

If the contingency keys are an inlist, it will be an inlist file that contains one or more contingency keys.

How can the contents of an X.509 certificate file be determined?

The PKQRYCDB commands designed to read and report on an end-entity X.509 certificate files and P7B containers. This command with public key files in CER format (either DER or Base64 encoded), private key files in PFX or P12 format (either DER or Base64 encoded) and PKCS#7 format files. See PKQRYCDB for more information.

```
                    Smartcrypt Query Cert Db   (PKQRYCDB)

Type choices, press Enter.

Processing Type  . . . . . . . . > *ALL          *SUMMARY, *LEVEL1, *ALL...
File Type  . . . . . . . . . . . > *FILE         *DB, *FILE, *P7B
Certificate Type . . . . . . . . > *PUBLIC       *PUBLIC, *PRIVATE, *ALL
Selection Name . . . . . . . . . >
'/yourpath/PKWARE/Cstores/public/bill_somebody2003
.cer'

Cert Passphrase  . . . . . . . . .
Logging Level  . . . . . . . . .   *LOG          *NOLOG, *LOG, *MAXLOG
```

The following is the resulting output of the job above, detailing the end-entity certificate information.

```
PKQRYCDB QUERY Smartcrypt Cert DataBase starting------2016/04/21 13:07:29
--- Certificate 1 ---
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:
   3FDF2A91 2B5A9F9B 46E0D8A0 96DBD04B
NotBefore:
   Mon Jul 21 20:00:00 2009
NotAfter:
   Wed Jul 21 19:59:59 2010
SHA-1 Hash of Certificate(Thumbprint):
   D5 CE FF A5 32 EF B6 53 EA 75 F7 CA 2E 01 85 7B 65 7C A8 E7
Public Key Hash:
   6E 16 CF EF FA A0 93 24 2B 89 DE E6 23 C7 D7 42 80 82 F3 E3
End Entity
```

```
PKQRYCDB Run Totals for Library PKW14961S:
   Total Records In Error  =0
   Total Records Processed =1
PKQRYCDB Scan ending------
```

You may also report on an intermediate CA, trust root CA, and/or a CRL by using the PKSTORADM command.

Here you will enter the store type in question and select RUNTYPE(*BROWSE) or RUNTYPE(*LIST). This option displays details about each certificate in the source file in a BROWSE window. From here you can determine the contents.

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.

How do I encrypt a file into an OpenPGP Format?

Creating an OpenPGP archive is basically the same as creating and encrypting files to a ZIP archive as described in Chapter 1 (Getting Started) of the User's Guide except using the PKPGPZ command with some modifications to the available options.

6 PKCFGSEC “Configure Security Parameters” Command

Requires Smartcrypt

PKCFGSEC Command Summary with Parameter Keyword Format

The PKCFGSEC command updates and views the Smartcrypt security configuration file. This configuration file is where the enterprise security options and defaults reside.

Keywords are demarcated by spaces. In many case 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.

Usage Notes

It is recommended that you define your standard enterprise options in a CL using the PKCFGSEC command. Then as changes occur or new versions are upgraded, the CL can be run to reset the parameters. This will also help if you need to temporarily change one or more settings, as the CL can provide a quick means to reset the configuration back to the enterprise setting.

```
TYPE (     ({*INSTALL}                 )
              {IVIEW }

SECTYPE (    ZIP Secure Settings:                )
               Smartcrypt Active  {*SAME|*NO|*YES}
               AES 3DES Keys {*SAME|*NO|*YES }
               Passphrase {*SAME|*NO|*RQD}
               Recipient Secure {*SAME|*NO|*RQD }
               File Signing {*SAME|*NO }                [For future release]
               Archive Signing {*SAME|*NO}              [For future release]
               File Authentication {*SAME|*NO }         [For future release]
               Archive Authentication {*SAME|*NO}       [For future release]
               File Name Encryption {*SAME|*NO|*RQD| *OPT}

ADVCRYPT (       Freeze Method                   )
                       {*SAME }
                       {*NONE }
```

```
                        {AES}
                        {AES128}
                        {AES192}
                        {AES256}
                        {3DES }
                        {DES }
                        {RC4_128}
                        {ADVANCE}
               Hash     {*SAME }
                        {*SHA1 }


PASSWORD (     Min Length {1-260}                )
               Min Length {1-260}
               Hardening Options{0}[1]


SIGNPOL  (  Signing Settings:                    )
               Signing Hash          {*SAME }
                                     {*SHA1}
                                     {*MD5}
                                     {*SHA256}
                                     {*SHA384}
                                     {*SHA512}

               Validate Level        {*SAME }
                                     {*VALIDATE}
                                     {*WARN }
                                     {*VALIDATELOCK}
                                     {*WARNLOCK}
               Lockdown              {*SAME }
                                     {*NO}
                                     {*YES}
               Filters               {*SAME }
                                     {*ALL}
                                     {*NONE}
                                     {*TRUSTED}
                                     {*EXPIRED}
                                     {*REVOKED}
                                     {*NOTTRUSTED}
                                     {*NOTEXPIRED}
                                     {*NOTREVOKED}


AUTHPOL   (  Authenticate Filters:                        )
               Validate Level        {*SAME }
                                     {*NONE }
                                     {*VALIDATE}
                                     {*WARN }
                                     {*VALIDATELOCK}
                                     {*WARNLOCK}
               Lockdown              {*SAME }
                                     {*NO}
                                     {*YES}
               Filters               {*SAME }
                                     {*ALL}
                                     {*NONE}
                                     {*TAMPER}
                                     {*TRUSTED}
                                     {*EXPIRED}
                                     {*REVOKED}
                                     {*NOTAMPER}
                                     {*NOTTRUSTED}
                                     {*NOTEXPIRED}
                                     {*NOTREVOKED}

ENCRYPOL (  Encryption Filters:                  )
               Validate Level        {*SAME }
                                     {*NONE }
                                     {*VALIDATE}
```

```
                                  {*WARN }
                                  {*VALIDATELOCK}
                                  {*WARNLOCK}
               Lockdown           {*SAME }
                                  {*NO}
                                  {*YES}
               Filters            {*SAME }
                                  {*ALL}
                                  {*NONE}
                                  {*TRUSTED}
                                  {*EXPIRED}
                                  {*REVOKED}
                                  {*NOTTRUSTED}
                                  {*NOTEXPIRED}
                                  {*NOTREVOKED}


DECRYPOL (  Decryption Filters:                ) [1]
               Validate Level     {*SAME }

REVKEPOL (  Revocation Settings:               )
               CRL Type           {*SAME }
                                  {*NONE }
                                  {*STATICCRL}
               Revoke Level       {*SAME } [1]
                                  {*NONE}


FACENC   (  Encryption Facilities:             )
               Lockdown           {*SAME }
                                  {*NO}
                                  {*YES}
               Fac. Type          {*SAME }
                                  {*NONE}
                                  {PKSW}
                                  {IBMSW}


FACHASH  (  Hashing Facilities:                )
               Lockdown           {*SAME }
                                  {*NO}
                                  {*YES}
               Fac. Type          {*SAME }
                                  {*NONE}
                                  {PKSW}
                                  {IBMSW}



CERTSELT (  Certificate Validation Level:              )
               Cert. Best Fit              {*SAME }
                                           {*NONE }
                                           {*LATESTVALID}
                                           {*LASTVALID}
                                           {*FIRSTVALID}
                                           {*LATEST}
                                           {*LAST }
                                           {*FIRST }
               Cert. Secure Type{*SAME } [1]
                                           {*NONE }
                                           {*ENCRYPTION}
                                           {*SIGNING}


CERTDB (Certificate DataBase Locator:                  )
                         Active?           {*SAME }
                                              {*YES }
                                               {*NO }
```

```
                          Store Type        {*SAME }
                                                {*NONE }
                                                {*LIB}
                          Sequence          {*SAME }
                                                {0-9 }
                          Library           {*SAME }
                                                {Library Name string }
                          Search Mode       {*SAME }
                                                {*EMAIL }
                                                {*CN}

CSPUB (  Certificate PUBLIC Store:                        )
                          Store Type        {*SAME }
                                                {*NONE }
                                                {*PATH}
                          Sequence          {*SAME }
                                                {0-9 }
                          Store Location    {*SAME }
                                                {Path Name string }


CSPRVT (  Certificate PRIVATE Store:                      )
                          Store Type        {*SAME }
                                                {*NONE }
                                                {*PATH}
                          Sequence          {*SAME }
                                                {0-9 }
                          Store Location    {*SAME }
                                                 {Path Name string }


CSCA (   Certificate CA Store:                    )
                          Store Type        { *SAME }
                                                {*NONE }
                                                {*FILE}
                          Sequence          { *SAME }
                                                {0-9 }
                          Store Location    { *SAME }
                                                {Path/File Name string }


CSROOT (   Certificate ROOT Store:                        )
                          Store Type        { *SAME }
                                            {*NONE }
                                            {*FILE}
                          Sequence          { *SAME }
                                                {0-9 }
                          Store Location    { *SAME }
                                                {Path/File Name string }

CSCRL  (   Certificate CRL Store:                 )
                          Store Type        { *SAME }
                                            {*NONE }
                                            {*FILE}
                          Sequence          { *SAME }
                                                {0-9 }
                          Store Location    { *SAME }
                                                {Path/File Name string }


CWRKPATH (   Certificate Work Path:                       )
                          Path Location     { *SAME }
                                            {*NONE }
                                            {Path Name string }

PLPATH (   Smartcrypt Partner Path:                               )
                          Path Location     {*SAME }
                                             {*NONE }
```

```
                                        {Path Name string }


ENTPREC (  Contingency Key(s):                  )
           LookUp Type                  { *SAME }
                                         {*NONE}
                                         {*DB }
                                         {*LDAP }
                                         {*FILE }
                                         {*MBRSET }
                                         {*INLISTDB }
                                         {*INLISTIFS }
                                         {*PGPDEF }

           Recipient                     {Recipient Name string }
           Required                      { *SAME }
                                         {*RQD }
                                         {*OPT }
PGPRULE (  OpenPGP Definitions:                 )
           Allow Keys for Smartcrypt     { *SAME }
                                         {*YES}
                                        {*NO }

           Allow PKPGPZ (ZIP)           { *SAME }
                                         {*YES}
                                         {*NO }

           Allow PKPGPU (UNZIP)         { *SAME }
                                         {*YES}
                                         {*NO }

           Encryption Key Select        { *SAME }
                                         {*NONE}
                                         {*LATESTVALID}
                                         {*LASTVALID}
                                         {*FIRSTVALID}
                                         {*FIRST}
                                         {*LAST}
                                         {*LATEST}

           Signing Key Select           { *SAME }
                                         {*NONE}
                                         {*LATESTVALID}
                                         {*LASTVALID}
                                         {*FIRSTVALID}
                                         {*FIRST}
                                         {*LAST}
                                         {*LATEST}

           RFC2440 Level                 { *SAME }
                                         {*NONE}
                                         {*LEVEL1}


PGPKEYPUB  (   OpenPGP PUB Keyring:             )
           Keyring Handle               { *SAME }
                                         {Keyring Handle string name }

           Keyring Engine              { *SAME }
                                        {*NONE }
                                        {*FILE}

           Engine Name                 { *SAME }
                                        {Path/Keyring Name string }


PGPKEYPVT  (   OpenPGP PVT Keyring:             )
           Keyring Handle              { *SAME }
                                         {Keyring Handle string name }
```

```
               Keyring Engine              { *SAME }
                                            {*NONE }
                                            {*FILE}

               Engine Name                 { *SAME }
                                            {Path/Keyring Name string }


PGPPREC (  PGP Contingency Key(s):                  )
               LookUp Type                 { *SAME }
                                            {*NONE}
                                            {*PGPDEF}

               Recipient                   {Recipient Name string }
               Required                    { *SAME }
                                            {*RQD }
                                            {*OPT }


LDAPACTV (  LDAP Definitions:                       )
               Nbr Active LDAPs             { *SAME }
                                              {Number of Active LDAPs }


LDAP1 (Primary LDAP:                     )
               1 Network Name                { *SAME }
                                              {Network IP address or server Name }
               1 Port                       {1-9999 }
               1 Sequence                   {0-9 }
               1 TimeOut                    {0-9999 }
               1 Access User                {LDAP User for Access }
               1 Access Passphrase          {User Passphrase }
               1 Search Mode                {*EMAIL }
                                            {*CN }
               1 Start Node                 {Start Node Text String }

               1 Test                       {N }
                                            {Y }

NSSRULES (  NSS Process Settings:                   )
               NSS Required                 {*SAME }
                                            {*NO }
                                            {*OPT}
                                            {*REQ}
               NSS Classify Archive         {*SAME }
                                            {INACTIVE }
                                            {SECRET_SUITEB_REQPLUS}
                                            {SECRET_SUITEB_STRICT}
                                            {TOPSECRET_SUITEB_REQPLUS}
                                            {TOPSECRET_SUITEB_STRICT}
               NSS Check Archive State      {*SAME }
                                            {*NO }
                                            {*WARN}
                                            {*FAIL}
```

PKCFGSEC Command Keyword Details

When you use the *SAME option, no changes take place to the current security configuration file.

TYPE

TYPE(*UPDATE|*VIEW)

The type processing parameter indicates whether the security configuration should update the configuration file or view the current settings.

SECTYPE

```
ZIP Secure Settings:
      Smartcrypt Active  . . . . .  *SAME         *NO, *YES, *SAME
     AES 3DES Keys . . . . . . . .   *SAME         *NO, *YES, *SAME
     Passphrase  . . . . . . . . .   *SAME         *NO, *RQD, *SAME
     Recipient Secure   . . . .   *SAME         *NO, *RQD, *SAME
     File Signing  . . . . . . .   *SAME         *NO, *YES, *SAME
     Archive Signing   . . . . .   *SAME         *NO, *YES, *SAME
     File Authentication   . . .   *SAME         *NO, *YES, *SAME
     Archive Authentication  . .   *SAME         *NO, *YES, *SAME
     File Name Encryption  . . .   *SAME         *NO, *RQD, *OPT, *SAME
```

Or:

SECTYPE(*YES *YES *NO *NO *NO *NO *NO *OPT)

SECTYPE or PKZIP Secure Settings are the base security settings for ZIP and UNZIP processing and consist of six options:

Smartcrypt Active (*NO|*YES|*SAME)

Setting this option to *YES activates the security configuration file. This is required to utilize the security features in Smartcrypt.

AES 3DES Keys (*NO|*YES|*SAME)

The purpose of this switch is to maintain compatibility with Windows (pre-XP) systems where the private key certificate was not imported with "Mark the private key as exportable". This has importance when sharing AES-encrypted files with recipients. See the section on Windows compatibility for more information.

Passphrase (*NO|*RQD|*SAME)

The Passphrase option, if coded *YES, will force the ZIP process to always require a passphrase when compressing a file.

Recipient Secure (*NO|*RQD|*SAME)

The Recipient Secure option, if coded *YES, will force the ZIP process to always require at least one valid recipient when compressing a file.

File Signing (*NO |*YES |*SAME)

Allows Smartcrypt to digitally sign the files in the archive.

Archive Signing (*NO |*YES |*SAME)

Allows Smartcrypt to digitally sign the archive's central directory.

File Authentication (*NO |*YES |*SAME)

Allows Smartcrypt to authenticate the files in the archive that has been digitally signed.

Archive Authentication (*NO |*YES |*SAME)

Allows Smartcrypt to authenticate the archive's directory that has been digitally signed.

File Name Encryption (*NO |*RQD |*OPT |*SAME)

The file name encryption option allows the user to mask the names of files in the archive for added security.

> ***NO** – The ZIP process is not allowed to create file name-encrypted archives (that is, FNE(*NO) is required).
>
> ***RQD** – All ZIP runs must specify the FNE(*YES) to create file name-encrypted archives.
>
> ***OPT** – Creating file name-encrypted archives is optional for ZIP runs (that is, FNE(*YES) is allowed in the ZIP jobs).

ADVCRYPT

```
Strong Encryption:
       Lockdown Method . . . . .   *SAME          *SAME, *NONE, AES, AES128..
       Hash  . . . . . . . . . .   *SAME          *SAME, *SHA1
```

Or:

> **ADVCRYPT(AES128 *SHA1)**

The Advanced Encryption parameter controls the enterprise settings for encryption when zipping or compressing files. There are the following option settings for ADVCRYPT:

Method (AES|AES128|AES192|AES256|3DES|DES RC4_128| Advance|*SAME|*NONE)

The Method option, if coded other than *NONE, will force the ZIP process to always require the encryption algorithm selected. If AES is selected, then all AES algorithms are allowed. If Advance is selected, then encryption is required with any algorithm except ZIPSTD.

Hash (*SHA1 |*SAME)

A hash calculation is involved as part of the key manipulation for the encryption process. At this time only the SHA1 algorithm is supported.

PASSWORD

```
Archive Passphrase Specs:
     Min Length  . . . . . > 1            1-260, *SAME
     Max Length  . . . . . > 200          1-260, *SAME
     Hardening Options . . . . > 0        *SAME, 0, 1, 2
```

Or:

PASSWORD(1 260 0) or PASSWORD(1 260)

The Password parameter defines the enterprise settings when the ZIP process provides a passphrase. There are three options.

Min Length (1-260|*SAME)

The minimum length option is the enterprises required minimum length when a passphrase is provided in the ZIP process. The default is 1. The Windows version requires the minimum passphrase length to be 8 or more.

Max Length (1-260|*SAME)

The maximum length option is the enterprises required maximum length when a passphrase is provided in the ZIP process. The default is 260.

Hardening Options (0,1,2|*SAME)

The passphrase hardening option is reserved for future releases of Smartcrypt.

SIGNPOL

```
Signing Settings:
    Signing Hash . . . . . .   *SAME        *SAME, *SHA1, *MD5...
    Validate Level     . . . .   *SAME
       Lockdown Filters  . . .   *SAME        *NO, *YES, *SAME
       Filters . . . . . .   *SAME        *SAME, *ALL, *NONE...
              + for more values
```

Or

SIGNPOL (*SHA1 *WARN *NO (*ALL *NOTREVOKED))
SIGNPOL (*SHA1 *VALIDATELOCK *YES (*ALL *REVOKED))

When signing a file or archive, SIGNPOL will define the HASH ALGORITHM, the error validation levels, and the CERTIFICATE filters for the signers' certificates selected.

Signing Hash (*SHA1 | *MD5 |*SAME | *SHA256 | *SHA384 | *SHA512)

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.

Options:

***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.

The information below is from FIPS 180-1:

> This Standard specifies a Secure Hash Algorithm, SHA-1, for computing a condensed representation of a message or a data file. When a message of any length $< 2^{64}$ bits is input, the SHA-1 produces a 160-bit output called a message digest. The message digest can then be input to the Digital Signature Algorithm (DSA) which generates or verifies the signature for the message. Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller in size than the message. The same hash algorithm must be used by the verifier of a digital signature as was used by the creator of the digital signature.
>
> The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify.

The information below relates to FIPS 180-2:

> SUMMARY: The Secretary of Commerce has approved FIPS 180-2, Secure Hash Standard, and has determined that the standard is compulsory and binding on Federal agencies for the protection of sensitive, unclassified information.
>
> FIPS 180-2, Secure Hash Standard, replaces FIPS 180-1, which was issued in 1992 and which specified an algorithm (SHA-1) for producing a 160-bit output called a message digest. The message digest is a condensed representation of electronic data and is used in cryptographic processes such as digital signatures and message authentication. FIPS 180-2 includes three additional algorithms, which produce 256-bit, 384-bit, and 512-bit message digests. These expanded capabilities are compatible with and support the strengthened security requirements of FIPS 197, Advanced Encryption Standard.
>
> EFFECTIVE DATE: This standard is effective February 1, 2003.

Specifications: FIPS 180-2 is available on the NIST web page at: http://csrc.nist.gov/encryption/tkhash.html.

Processing Notes

When selecting a HASH algorithm, ensure that the partner product intended to complete the authentication of the digital signature supports the specified algorithm.

The entire data stream (archive central directory or file data) 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.

During authentication operations, ***Smartcrypt for IBM i*** will dynamically detect which algorithms had been used for signing and perform the necessary processing to validate the signature.

In general, a more secure the hashing method uses more CPU. One exception is the SHA512 hash, which takes about the same resources as the SHA384 hash, since SHA384 is a subset of SHA512. The PKCRYRUN utility command (see Chapter 12) can help you analyze CPU usage.

Validation Level (*VALIDATE | *WARN | *VALIDATELOCK | *WARNLOCK | *SAME)

The validation level defines the error level for failure, if an error occurs with the selection of a certificate for signing fails the filters.

Options:

***VALIDATE**	If an error occurs then the ZIP job will report that a failure occurred and will not continue.
***WARN**	If an error occurs then the ZIP will continue and will not report that an error occurred at the end of the ZIP process.
***VALIDATELOCK**	Same as *VALIDATE, but will also not allow a ZIP to run with an option for SIGNPOL other than what is defined for the enterprise system. A warning message will be issued and the run will continue using the enterprise settings.
***WARNLOCK**	Same as *WARN, but will also not allow a ZIP to run with an option for SIGNPOL other than what is defined for the enterprise system. A warning message will be issued and the run will continue using the enterprise settings.

Lockdown Filters (*NO | *YES | *SAME)

Provides the ability to lockdown the filters so that they cannot be changed at run time.

Specifying *YES puts the enterprise settings in lockdown mode. If the SIGNPOL in the PKZIP command is other than *SYSTEM, a warning message will be issued and the enterprise settings will be used for the run.

Filters (*ALL | *NONE | * TRUSTED | *EXPIRED | *REVOKED | *NOTTRUSTED | *NOTEXPIRED | *NOTREVOKED |*SAME)

The signing filter defines the level of processing that Signing Files and Signing Archives certificate selection will perform during SECZIP execution.

Options:

***ALL**	*TRUSTED, *EXPIRED, and *REVOKED are used.
***NONE**	*NOTTRUSTED, *NOTEXPIRED, and *NOTREVOKED are used
***TRUSTED**	Each end-certificate used in the signature must be traced back to a trusted root certificate. The CSCA 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 for the signing operation contains internal date ranges of validity. The certificate selection 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-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 signing certificate selection 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. REVOKED is not supported for OpenPGP files.
***NOTTRUSTED**	*TRUSTED is not validated for signing certificates.
***NOTEXPIRED**	*EXPIRED is not validated for signing certificates.
***NOTREVOKED**	*REVOKED is not validated for signing certificates.

AUTHPOL

```
Authenticate Filters:
    Validate Level     . . . . .   *SAME
       Lockdown Filters  . . . .   *SAME         *NO, *YES, *SAME
       Filters . . . . . . . . .   *SAME         *SAME, *ALL, *NONE...
               + for more values
```

Or:

AUTHPOL (*WARN *NO (*ALL *NOTREVOKED))
AUTHPOL (*VALIDATELOCK *YES (*ALL *REVOKED))

AUTHPOL defines the level of processing that authentication process will perform. AUTHPOL fully defines the signature authentication elements to be verified in both PKZIP and PKUNZIP. If not locked down the default settings may be changed by the Smartcrypt run.

Validation Level (*VALIDATE | *WARN | *NONE | *VALIDATELOCK | *WARNLOCK |*SAME)

The validation level defines the error level for failure, if an error occurs with the selection of a certificate for authenticating fails the filters.

Options:

***VALIDATE**	If an error occurs, then the ZIP job will issue that a failure occurred and will not continue.
***WARN**	If an error occurs, then the ZIP will continue and will not issue that an error occurred at the end of the ZIP process.
***NONE**	If an error occurs then the ZIP will continue and will not issue that an error occurred at the end of the ZIP process.
***VALIDATELOCK**	Same as *VALIDATE, but will also not allow a ZIP to run with an option for AUTHPOL other than what is defined for the enterprise system. A warning message will be issued and the run will continue using the enterprise settings.
***WARNLOCK**	Same as *WARN, but will also not allow a ZIP to run with an option for AUTHPOL other than what is defined for the enterprise system. A warning message will be issued and the run will continue using the enterprise settings.

Lockdown Filters (*NO | *YES | *SAME)

Provides the ability to lockdown the filters so they cannot be changed at run time.

By specifying *YES, the enterprise settings will be in lockdown mode. If the AUTHPOL in the PKZIP command is other than *SYSTEM, a warning message will be issued and the enterprise settings will be used for the run.

Filters (*ALL | *NONE | *TAMPER | *TRUSTED | *EXPIRED | *REVOKED | *NOTAMPER | *NOTTRUSTED | *NOTEXPIRED | *NOTREVOKED |*SAME)

The authentication filter defines the level of processing that authenticating Files and authenticating Archives certificate selection will perform during SECZIP execution.

Options:

***ALL**	*TAMPER, *TRUSTED, *EXPIRED, and *REVOKED are used.
***NONE**	*NOTAMPER, *NOTTRUSTED, *NOTEXPIRED, and *NOTREVOKED are used.
***TAMPER**	The signature associated with the archive or file(s) involved will be used to verify that the content has not been altered since the archive was built.
***TRUSTED**	Each end-certificate used in the signature must be traced back to a trusted root certificate. The CSCA 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 for the signing operation contains internal date ranges of validity. The certificate selection 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-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 signing certificate selection 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. REVOKED is not supported for OpenPGP files.
***NOTAMPER**	The signature associated with the archive or file(s) involved will NOT verify.
***NOTTRUSTED**	*TRUSTED is not validated for signing certificates.
***NOTEXPIRED**	*EXPIRED is not validated for signing certificates.
***NOTREVOKED**	*REVOKED is not validated for signing certificates.

ENCRYPOL

```
Encryption Filters:
    Validate Level     . . . . .   *SAME
       Lockdown Filters  . . . .   *SAME         *NO, *YES, *SAME
       Filters . . . . . . . . .   *SAME         *SAME, *ALL, *NONE...
               + for more values
```

Or:

ENCRYPOL (*WARN *NO (*ALL *NOTREVOKED))
ENCRYPOL (*VALIDATELOCK *YES (*ALL *REVOKED))

When encrypting a file with digital certificate, the certificates must meet a certain standard. ENCRYPOL defines the error validation levels and the certificate filters for the encrypting certificates selected.

Validation Level (*VALIDATE | *WARN | *VALIDATELOCK | *WARNLOCK | *NONE |*SAME)

The validation level defines the error level for failure, if an error occurs with the selection of a certificate for signing fails the filters.

Options:

***VALIDATE**	If an error occurs, then the ZIP job will issue that a failure occurred and will not continue.
***WARN**	If an error occurs, then the ZIP will continue and will not issue that an error occurred at the end of the ZIP process.
***VALIDATELOCK**	Same as *VALIDATE, but will also not allow a ZIP to run with an option for ENCRYPOL other than what is defined for the enterprise system. A warning message will be issued and the run will continue using the enterprise settings.
***WARNLOCK**	Same as *WARN, but will also not allow a ZIP to run with an option for ENCRYPOL other than what is defined for the enterprise system. A warning message will be issued and the run will continue using the enterprise settings.
***NONE**	No level system setting is set.

Lockdown Filters (*NO | *YES | *SAME)

Provides the ability to lockdown the filters so they cannot be changed at run time.

By specifying *YES, the enterprise settings will be in lockdown mode. If the ENCRYPOL in the PKZIP command is other than *SYSTEM, a warning message will be issued and the enterprise settings will be used for the run.

Filters (*ALL | *NONE | * TRUSTED | *EXPIRED | *REVOKED | *NOTTRUSTED | *NOTEXPIRED | *NOTREVOKED |*SAME)

The encryption filters defines the level of processing that encrypting certificate selection will perform during SECZIP execution.

Options:

***ALL**	*TRUSTED, *EXPIRED, and *REVOKED are used.
***NONE**	*NOTTRUSTED, *NOTEXPIRED, and *NOTREVOKED are used
***TRUSTED**	Each end-certificate used in the signature must be traced back to a trusted root certificate. The CSCA 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 for the signing operation contains internal date ranges of validity. The certificate selection 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-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 signing certificate selection 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. REVOKED is not supported for OpenPGP files.

***NOTTRUSTED** *TRUSTED is not validated for encrypting certificates.

***NOTEXPIRED** *EXPIRED is not validated for encrypting certificates.

***NOTREVOKED** *REVOKED is not validated for encrypting certificates.

REVKEPOL

```
Revocation Settings:
     CRL Type           . . . . .   *SAME         *STATICCRL, *NONE, *SAME
     Revoke Level     . . . . . .   *SAME         *SAME, *NONE
```

Or:

REVKEPOL (*WARN *NO (*ALL *NOTREVOKED))
REVKEPOL (*VALIDATELOCK *YES (*ALL *REVOKED))

REVKEPOL defines the type and level of CRL processing, if any, that will take place. Currently only a static CRL is available.

CRL Type (*STATICCRL | *NONE |*SAME)

Defines type CRL processing that will take place with certificate analysis.

Options:

***STATICCRL** Specifies CRL processing only on a static store that is updated with the PKSTORADM command. The timeliness of updates depends on the customer's policies and the updating of the CRL store. This option requires that the CRL must be defined.

***NONE** No CRL processing check will take place. With the option *NONE, no certificates will ever have the revoke status.

Revoke Level (*NONE | *SAME)

Currently not used. Reserved for future use.

***NONE** To be defined.

FACENC

Requires Smartcrypt with IBM i OS V5R3M0 or above

```
Encryption Facilities:
    Lockdown . . . . . . . . . .   *SAME          *NO, *YES, *SAME
    Fac. Type  . . . . . . . . .   *SAME          *SAME, *NONE, PKSW...
               + for more values
```

Or:

FACENC (*SAME (PKSW))
FACENC (*YES (IBMSW PKSW))

FACENC defines the encryption algorithm APIs that are available and their sequences. There are only two facilities of APIs: PKWARE, and IBM Software Security.

There are two entries for the FACENC parameter:

Lockdown (*NO | *YES | *SAME)

Provides the ability to lockdown the facilities used for encryption algorithms so they cannot be changed at run time.

Specifying *YES places the enterprise settings in lockdown mode, causing PKZIP and PKUNZIP overrides to be ignored.

Fac. Type (*NONE | PKSW | IBMSW |*SAME)

The encryption facility API Types and in their priority sequence. If both PKWSW and IBMSW will be used, specify the types in the priority of use. For example: FACENC (*SAME (IBMSW PKSW))

Options:

***NONE**	No encryption facility defined. Defaults to the distributed API, which is PKWARE.
PKSW	PKWARE Encryption APIs
IBMSW	IBM Encryption APIs

FACHASH

Requires Smartcrypt with IBM i OS V5R3M0 or above

```
Hashing Facilities:
    Lockdown . . . . . .   *SAME          *NO, *YES, *SAME
    Fac. Type  . . . . . .   *SAME          *SAME, *NONE, PKSW...
               + for more values
```

Or:

FACHASH (*SAME (PKSW))
FACHASH (*YES (IBMSW PKSW))

FACHASH defines the hashing algorithm APIs that are available and their sequences. There are only two facilities of APIs: PKWARE and IBM Software Security.

There are two entries for the FACENC parameter:

Lockdown (*NO | *YES | *SAME)

Provides the ability to lockdown the facilities used for hashing algorithms so they cannot be changed at run time.

Specifying *YES places the enterprise settings in lockdown mode, causing PKZIP and PKUNZIP overrides with the FACILITY parameter to be ignored.

Fac. Type (*NONE | PKSW | IBMSW |*SAME)

The hashing facility API Types in their priority sequence. If both PKWSW and IBMSW will be used, specify the types in the priority of use. For example: FACHASH (*SAME (IBMSW PKSW))

Options:

***NONE**	No hashing facility defined. Defaults to the distributed API, which is PKWARE.
PKSW	PKWARE Hashing APIs
IBMSW	IBM Hashing APIs

CERTSELT

```
Certificate Validation Level:
    Cert. Best Fit . . . . . > *NONE          *NONE, *LATESTVALID...
    Cert. Secure Type  . . . . . > *NONE          *NONE, *ENCRYPTION...
```

Or:

CERTSELT(*NONE *NONE)

The certificate validation level defines how Smartcrypt will select a digital certificate when there is more than one certificate available from an LDAP or database selection. There are two options.

Cert. Best Fit
(*NONE|*LATESTVALID|*LASTVALID|*FIRSTVALID|*FIRST|*LAST|*LATEST|*SAME)

This command assists in restricting the number or type of certificates being used to represent a user or organization for each encrypted file.

When using LDAP or the database to locate public-key certificates for recipients, it is possible to locate more than one certificate for a target recipient. For example, if a user obtains a new certificate each year, then multiple certificates may represent that user within the LDAP. It may also be possible for a user to have certificates from multiple certificate authorities (e.g., VeriSign, Thawte), or multiple certificates for different purposes (encryption vs. signing).

In any of the above conditions, a ZIP process may result in multiple recipient certificates being processed for the same target recipient (person or organization). Some organizations may desire to restrict the type or quantity of certificates being used for encryption. This can save processing resources and ZIP archive space.

Options:

***NONE**	No matching will be performed. Every certificate located in an LDAP server or database matching the search criteria will be added as a viable recipient.
***FIRST**	For each LDAP or database entry matching the search criteria, only the first certificate stored in that entry will be included, regardless of use type designated in the certificate or its valid date period. This use case depends on the certificate loading order used in the LDAP or database.
***LAST**	For each LDAP or database entry matching the search criteria, only the last certificate stored in that entry will be included, regardless of use type designated in the certificate or its valid date period. This use case depends on the certificate loading order used in the LDAP or Database.
***LATEST**	For each LDAP or database entry matching the search criteria, the most recent certificate stored in that entry will be included, regardless of use type designated in the certificate. Note that if multiple certificates are found within an LDAP entry, certificates with their validity period not yet starting will be excluded unless they are the only certificates within the entry. In that case, the first certificate found will be selected.
***FIRSTVALID**	For each LDAP or database entry matching the search criteria, the first certificate found with a use type set for encryption stored in that entry will be included, regardless of its valid date period. This use case depends on the certificate loading order used in the LDAP. If no entries are found with the use type set to encryption, then the first certificate found in the LDAP or database entry will be selected.

***LASTVALID** For each LDAP or database entry matching the search criteria, the last certificate found with a use type set for encryption stored in that entry will be included, regardless of its valid date period. This use case depends on the certificate loading order used in the LDAP. If no entries are found with the use type set to encryption, then the first certificate found in the LDAP or database entry will be selected.

***LATESTVALID** For each LDAP entry matching the search criteria, the most recent certificate found with a use type set for encryption also having the "best" date range for its validity period. This use case depends on the certificate loading order used in the LDAP. If no entries are found with the use type set to encryption, then the most recent certificate found in the LDAP entry will be selected. Note that if multiple certificates are found within an LDAP entry, certificates with their validity period not yet starting will be excluded unless they are the only certificates within the entry. In that case, the first certificate found will be selected.

Note: Regardless of the option selected, at least one certificate will be selected from an LDAP or database entry. Each certificate selected must be in valid X.509 public-key format.

Cert. Secure Type (*NONE| *ENCRYPTION| *SIGNING| |*SAME)

The certificate secure type forces the strict enforcement of digital certificate based upon on their purpose. An example would be if a certificate was for signing only and did not allow encryption then by specifying *ENCRYPTION, the certificate would not be valid.

CERTDB

```
Certificate DataBase Locator:
     Active? . . . . . > *YES          *NO, *YES, *SAME
     Store Type  . . . . . > *LIB      *LIB, *SAME
     Sequence  . . . . . > 0           0-9
     Library . . . . . > PKW14961S     DB Library
     Search Mode . . . . . > *EMAIL    *EMAIL, *CN, *SAME
```

Or:

CERTDB(*YES *LIB 0 PKW14961S *EMAIL)

The certificate locator database parameter activates and defines the location and defaults of the certificate locator database files. There are five options.

Active? (*YES|*NO|*SAME)

To utilize the certificate locator database you must set this option to *YES. If this is *NO then all attempts to use the *DB option on recipients parameter in ZIP and UNZIP will fail.

Store Type (*LIB|*SAME)

The store type specifies what type of database store. At this time only *LIB for library is allowed.

Sequence (1-9*SAME)

The sequence defines the sequence of the database search when using the *SYSTEM option. This is reserved for future releases

Library (Library name|*SAME)

This defines where the database store exists and is used in combination with the "Store Type". At this time only *LIB is allowed, so this option will define the library where the Smartcrypt databases exist.

Search Mode Default (*EMAIL|*CN|*SAME)

The Search Mode default defines what type of search should be used when performing a database search and the search string prefix is not 'CN=' nor 'EM='. *EMAIL will use the email address contained in the certificate and *CN will use the common name of the certificate.

CSPUB

```
Certificate PUBLIC Store:
     Store Type  . . . . . > *PATH         *PATH, *SAME
     Sequence  . . . . . > 1             0-9
     Store Location  . . . . . > '/yourpath/PKWARE/Cstores/public'
```

Or:

CSPUB(*PATH 1 '/yourpath/PKWARE/Cstores/public')

The certificate public store parameter defines the location and defaults for the public certificate store. A store location is required for certificate processing. There are three options.

Store Type (*PATH|*SAME)

The store type specifies the type of store. At this time only *PATH is allowed and defines the "store Location" as a path in the integrated file system (IFS).

Sequence (1-9|*SAME)

The Sequence option defines the sequence of the public search when using the *SYSTEM option. This is reserved for future releases.

Store Location (location name|*SAME)

This defines the IFS path where the certificate store for individual public-key X.509 certificates resides on the local system.

CSPRVT

```
Certificate PRIVATE Store:
     Store Type  . . . . . . . . > *PATH         *PATH, *SAME
     Sequence  . . . . . . . . . > 0            0-9
     Store Location  . . . . . . > '/yourpath/PKWARE/Cstores/private'
```

Or CSPRVT(*PATH 0 '/yourpath/PKWARE/Cstores/private')

The private certificate store parameter defines the location and defaults for the private certificate store. A store location is required for certificate processing. There are three options.

Store Type (*PATH|*SAME)

The store type specifies the type of store. At this time only *PATH is allowed and defines the “Store Location” as a path in the IFS.

Sequence (1-9 |*SAME)

The Sequence defines the sequence of the private search when using the *SYSTEM option. This is reserved for future releases.

Store Location (location name|*SAME)

This defines the IFS path where the certificate store for individual private-key X.509 certificates resides on the local system.

Usage Notes

Private keys require a passphrase to use or open.

CSCA

```
Certificate CA Store:
     Store Type  . . . . . . . . > *FILE         *FILE, *SAME
     Sequence  . . . . . . . . . > 0            0-9
     Store Location  . . . . . . > '/yourpath/PKWARE/Cstores/CA/pkwareCA.store'
```

Or:

CSCA(*FILE 0 '/yourpath/PKWARE/Cstores/CA/ pkwareCA.store')

The certificate authority store parameter defines the location and defaults for the certificate authority store. A certificate authority store location is required to validate the certificate for trust and certificate revocation. There are three options.

Store Type (*FILE|*SAME)

The store type specifies the type of store. At this time only *FILE is allowed to define the “Store Location” as a path/file in the IFS.

Sequence (1-9 |*SAME)

The Sequence defines the sequence of the private search when using the *SYSTEM option. This is reserved for future releases.

Store Location (location name|*SAME)

This defines the IFS full path and file name where the certificate authority store for public-key X.509 certificates resides on the local system. This store file is a file with a PKCS#7 format.

CSROOT

```
Certificate ROOT Store:
     Store Type  . . . . . . . . > *FILE         *FILE, *SAME
     Sequence  . . . . . . . . . > 0            0-9
     Store Location  . . . . . . >
'/yourpath/PKWARE/Cstores/Root/pkwareRoot.store'
```

Or:

CSROOT(*FILE 0 '/yourpath/PKWARE/Cstores/Root/ pkwareRoot.store')

The certificate root store parameter defines the location and defaults for the trusted root public store. A root store location is required to validate the certificate for trust and certificate revocation. There are three options.

Store Type (*FILE|*SAME)

The store type specifies the type of store. At this time only *FILE is allowed to define the “Store Location” as a path/file in the IFS.

Sequence (1-9 |*SAME)

The sequence defines the sequence for the private search when using the *SYSTEM option. This is reserved for future releases.

Store Location (location name|*SAME)

This defines the IFS full path and file name where the trusted root public store for trusted root public-key X.509 certificates resides on the local system. This store file is a file with a PKCS#7 format.

CSCRL

```
Certificate CRL Store:
     Store Type  . . . . . . . . > *FILE         *FILE, *SAME
     Sequence  . . . . . . . . . > 0            0-9
     Store Location  . . . . . . > '/yourpath/PKWARE/Cstores/CRL/pkwareCRL.store'
```

Or:

CSCRL(*FILE 0 '/yourpath/PKWARE/Cstores/ pkwareCRL.store')

The certificate CRL store parameter defines the location and defaults for the Certificate Revocation Store. A CRL store location is required to perform certificate revocation which is set in REVKEPOL parameter. Only a Static CRL is available and has three options.

Store Type (*FILE|*SAME)

The store type specifies the type of store. At this time only *FILE is allowed to define the “Store Location” as a path/file in the IFS.

Sequence (1-9 |*SAME)

The sequence defines the sequence of the private search when using the *SYSTEM option. This is reserved for future releases.

Store Location (location name|*SAME)

This defines the IFS full path and file name where the CRL (Certificate Revocation List) store resides on the local system. This store file is a file with a PKCS#7 format.

CWRKPATH

```
Certificate Work Path:
     Path Location  . . . . . > '/yourpath/PKWARE/Cstores/CTemp'
```

Or:

CWRKPATH ('/yourpath/PKWARE/Cstores/CTemp')

The certificate administration tools and certificate utility tools require the use of a temporary area in order to write temporary files during a process. CWRKPATH defines the path where the utilities can create and destroy temporary files. This area should give *PUBLIC full rights.

Path Location (Path name |*NONE |*SAME)

Specify the full path name for the certificate working path. If no changes are required, use *SAME. If the current path is to be cleared, use *NONE. Note by not defining a working path certain certificate utilities may not operate correctly.

PLPATH *(SecureZIP Partner only)*

```
SecureZIP Partner Path:
     Path Location  . . . . . > '/yourpath/PKWARE/Cstores/Sponsor'
```

Or:

PLPATH ('/yourpath/PKWARE/Cstores/Sponsor')

This path is required only if the PKWARE SecureZIP Partner program is used. The PKWARE SecureZIP Partner program will not function if no valid path is defined. When a path is specified in this parameter, PKCFGSEC validates that the required subfolders exist and creates any that are missing.

The required subfolders are listed in the table below.

Required Sponsor Subfolders

/Package	Folder where the Sponsor's issuing package is saved. When a sponsor's package is received, it should be copied or Ftp'd (binary) to the folder so it can be installed.
/Info	Contains an external XML file listing the installed sponsors and package settings.
/Auth	Contains for each sponsor a PKCS#7 file that contains a collection of end-entity certificates that SecureZIP Partner for IBM i Read mode is to recognize as valid signers for the sponsor. This store also contains any CA certificates that would be necessary to verify trust chains of the end-entity certificates. SecureZIP Partner for IBM i Read mode uses this file to verify that the central directory of an archive to be extracted is properly signed.
/Recip	Contains for each sponsor a PKCS#7 file containing a collection of end-entity certificates to be used as recipients when encrypting data for the sponsor. The files also contain any CA certificates necessary to verify the trust chains of the end-entity certificates. The files are internally signed by the special SecureZIP Partner for IBM i Write mode certificate.
/CA_Auth	Contains for each sponsor a PKCS#7 file containing CA certificates that can issue end-entity certificates that would be considered trusted by SecureZIP Partner for IBM i for extraction. This file is internally signed using the same certificate that was used to sign the normal end-entity authentication store so that it can be verified outside of the ZIP archive.
/Log	Contains the SecureZIP Partner_Install.log file used to time-stamp installation of each sponsor file.
/Temp	Used to install the partner Sponsor Distribution Package. It is cleared at the start and end of each installation, so no permanent files should be saved in this folder.

Each file in the **I**nfo, **A**uth, **R**ecip, and **C**AAuth folders will contains files with the Sponsor assigned 7 ID number with a prefix for each file type in the folders.

PKPLINKIN command will install a sponsor's package with the package name that is in the Package folder. If this is a replacement for a sponsor, it will replace all appropriate files in the all folders. If the sponsor files exist and the control date of the installed files is greater than that of the files being installed, a message is given that the install failed due to trying to install an older sponsor file. If you need to install an older sponsor package, all of the installed sponsor files should be deleted first.

PKPLINKLST command will list all or one of the installed sponsor files defined in the INFO path.

Path Location (Path name |*NONE |*SAME)

Specify the full path name for the PKWARE SecureZIP Partner path. If no changes are required, use *SAME. If the current path is to be cleared, use *NONE. **Note**: Without a SecureZIP Partner path, the SecureZIP partner program will not function.

ENTPREC

```
Contingency Key(s):
    LookUp Type  . . . . . > *MBRSET   *NONE, *DB, *LDAP, *FILE...
    Recipient  . . . . . > 'pkwareCertAdmin04.cer'
    Required . . . . . .   *SAME     *RQD, *OPT, *SAME
```

Or:

ENTPREC(*MBRSET 'pkwareCertAdmin04.cer' *RQD)

The contingency key parameter defines the enterprise or corporate defined recipient which should be included as a global or administrative access recipient. This enables the enterprise to decrypt and access the file(s) when other Recipients are no longer able or eligible.

The specification of this contingency key does *not* trigger encryption to take place during ZIP processing in the same way as – ENTPREC parameter in the ZIP command. However, once strong encryption is specified, the value of this parameter is implicitly included in the run as if a - ENTPREC command had been invoked. This parameter follows the syntax of the ENTPREC in the PKZIP and PKUNZIP commands.

To provide for multiple Contingency Keys, *INLISTDB and *INLISTIFS provides the ability to define files with a list of contingency key recipients that will be used for the Contingency Keys.

There are three options.

Lookup Type (*NONE |*DB |*LDAP |*FILE |*MBRSET |* INLISTDB |* INLISTIFS |*PGPDEF |*SAME)

The lookup type would be the type of recipient search that will be used for the recipient string.

***NONE** - Indicates no contingency key is supplied.

***DB** - The recipient string is defined to search using the certificate locator database to access the contingency key certificate.

***LDAP** - The recipient string is defined to search using the LDAP server to access the contingency key certificate.

***FILE** - The recipient string is defined to search using a specific path and file in the IFS to access the contingency key certificate.

***MBRSET** - The recipient string is defined to search using a file in the enterprise public certificate store to access the contingency key certificate.

***INLISTDB** - The recipient string is defined to be a file member in a library that contains all recipients for the contingency key. It has the format LIBRARY/FILE(MBR). For the format of the file see (Step 9: Using Input List File for ENTPREC Certificate List).

***INLISTIFS** - The recipient string is defined to be a stream file in the IFS that contains all recipients for the contingency key. It full path and file name. For the format of the file see (Step 9: Using Input List File for ENTPREC Certificate List).

***PGPDEF** - OpenPGP key access is defined in an OpenPGP keyring. The Recipient string will either be an Email address, Common name or Key ID for the OpenPGP key.

Recipient (The contingency key recipient file or name)

The common name (CN=) or email address (EM=) of the recipient(s). Also, the location on the Inlist file containing the recipient name(s). If this is *PGPDEF then the format is 'handle,search-key', where search-key is CN=, EM=, or KEYID=.

Required (*RQD | *OPT |*SAME)

Indicates whether the Recipient will fail the job, if the certificate cannot be loaded and is valid.

- *SAME - No changes to current settings.
- *RQD REQUIRED - Anytime the ENTPREC is evoked in the PKZIP command, the recipient defined here is required for the ZIP process to complete successfully.
- *OPT OPTIONAL - If the certificate cannot be loaded or is invalid the process will continue.

PGPRULE

```
OpenPGP Definitions:
   Allow Keys for Smartcrypt  . .   *SAME          *NO, *YES, *SAME
   Allow PKPGPZ (ZIP)  . . . . .   *SAME         *NO, *YES, *SAME
   Allow PKPGPU (UNZIP)  . . . .   *SAME         *NO, *YES, *SAME
   Encryption Key Select . . . .   *SAME         *NONE, *LATESTVALID...
   Signing Key Select  . . . . .   *SAME         *NONE, *LATESTVALID...
   RFC2440 Level       . . . . .   *SAME         *NONE, *LEVEL1, *SAME.
```

Or:

PGPRULE(*YES *YES *NO LATESTVALID LATESTVALID *NONE)

PGPRULE Secure Settings are the base security settings for OpenPGP processing and consist of six options:

Allow Keys for Smartcrypt (*NO|*YES|*SAME)

Setting this option to *YES activates the security configuration file, allowing you to use OpenPGP keyrings with Smartcrypt. OpenPGP keyrings are the equivalent of X.509 certificates, storing public and private keys. Activating the security configuration file is required to utilize the OpenPGP keyrings security features for Smartcrypt.

***NO** – Smartcrypt will be deactivated for use of OpenPGP keyrings in PKZIP and PKUNZIP.

***YES** – Smartcrypt will be activated for utilization of OpenPGP keyrings for encryption and decryption in PKZIP and PKUNZIP commands.

Allow PKPGPZ (ZIP) (*NO|*YES|*SAME)

Setting this option to *YES activates the use of the PKPGPZ command to create OpenPGP files. This is required to utilize the OpenPGP PKPGPZ command in Smartcrypt.

***NO** – PKPGPZ command will be deactivated for use.

***YES** – PKPGPZ command will be activated for creation of OpenPGP files.

Allow PKPGPU (UNZIP) (*NO|*YES|*SAME)

Setting this option to *YES activates the use of the PKPGPU command to extract OpenPGP files. This is required to utilize the OpenPGP PKPGPU command in Smartcrypt.

***NO** – PKPGPU command will be deactivated for use.

***YES** – PKPGPU command will be activated for extraction of OpenPGP files.

Encryption Key Select (*NONE|*LATESTVALID|*LASTVALID|*FIRSTVALID|*FIRST |*LAST|*LATEST|*SAME)

This option sets the system settings for PKPGPZ and PKPGPU parameter PGPRULES Encryption Key Select.

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.

***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.

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 timestamp 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 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 ENTPREC parameter to specify the precise key to be used.

Signing Key Select (*NONE|*LATESTVALID|*LASTVALID|*FIRSTVALID|*FIRST |*LAST|*LATEST|*SAME)

This option sets the system settings for PKPGPZ and 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 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.

***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.

***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.

***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.

***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.

***SYSTEM** Use the enterprise setting from PKCFGSEC parameter PGPRULE for "Signing Key Select".

Usage Note:

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/authentication.

When 'date range' is referred to, the specificity is actually a timestamp 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 SIGNERS parameter to specify the precise key to be used.

RFC2440 Level (*NONE |*LEVEL1 |*SAME)

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 specific amounts of backward compatibility to the RFC 2440 standards.

***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, this setting may be set for ZIP processing so that RFC 2440 encipherment packets (tag 9) are created instead of packet tag 18.

PGPKEYPUB OpenPGP PUB Keyring

```
OpenPGP PUB Key Ring:
    Key Ring Handle    . . . . .   *SAME          Handle, *NONE, *SAME
    Key Ring Engine    . . . . .   *FILE          *FILE
    Engine Name  . . . . . . . .   _______________
```

Or:

PGPKEYPUB('MYSYSPUB' *FILE '/mypast/syskeyring.pubring.pkr')

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 for each run.

There are three entries for the PGPKEYPUB parameter:

Keyring Handle (Handle|*NONE|*SAME)

Defines the global desired 'handle' name for the system public keyring.

Keyring Engine (*FILE)

Specifies the method to access OpenPGP keyring. Currently only *FILE is available.

Engine Name (path/file string name)

This defines the IFS full path and file name for the Public Keyring 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 public OpenPGP keyring.

PGPKEYPVT OpenPGP PVT Keyring

```
OpenPGP PVT Key Ring:
    Key Ring Handle    . . . . .   *SAME          Handle, *NONE, *SAME
    Key Ring Engine    . . . . .   *FILE          *FILE
    Engine Name  . . . . . . . .   _______________
```

Or:

PGPKEYPVT('MYSYSPVT' *FILE '/mypast/syskeyring.secring.skr ')

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.

There are three entries for the PGPKEYPVT parameter:

Keyring Handle (Handle|*NONE|*SAME)

Defines the global desired 'handle' name for the system private keyring.

Keyring Engine (*FILE)

Specifies the method to access OpenPGP keyring. Currently only *FILE is available.

Engine Name (path/file string name)

This defines the IFS full path and file name for the Private Keyring 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 private OpenPGP keyring.

PGPPREC

```
PGP Contingency Key(s):
    LookUp Type  . . . . . .   *SAME         *NONE, *SAME, *PGPDEF
    Handle       . . . . . .   *SAME         Handle, *NONE, *SAME
    Recipient  . . . . . . .
    Required . . . . . . . .   *SAME         *RQD, *OPT, *SAME
```

Or:

PGPPREC(*PGPDEF 'handle' 'EM=john.doe@pkware.com' *OPT)

PGPPREC(*PGPDEF 'handle' 'KEYID=1234567890123456' *RQD)

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. This enables the enterprise to decrypt and access the file(s) when other Recipients are no longer able or eligible. Specifying a contingency key does not trigger encryption to take place during PKPGPZ processing in the same way as –PGPPREC parameter in the ZIP command. However, once strong encryption is specified, the value of this parameter is implicitly included in the run as if a -PGPPREC command had been invoked. This parameter follows the syntax of the PGPDEF in the PKPGPZ and PKPGPU commands.

NOTE: Make sure that the proper keyring handle is included in the PKPGPZ run with PGPDEF or is defined in PGPKEYPUB.

There are four entries for the PGPPREC parameter:

Lookup Type (*NONE |*PGPDEF |*SAME)

The lookup type would be the type of recipient search that will be used for the recipient string.

***NONE** - Indicates no PGP contingency key is supplied.

***PGPDEF** - OpenPGP key access is defined in a Key definition (See PGPDEF). The Recipient string will either be an Email address, Common name or Key ID of the OpenPGP key.

Handle (The Keyring Handle string name)

The Keyring Handle (up to 8 bytes) is used to match up with the proper supplied keyring (See PGPDEF).

Recipient (The contingency key recipient file or name)

The common name (CN=), email address (EM=) or KEYID= of the recipient(s).

Required (*RQD | *OPT |*SAME)

Indicates whether the Recipient will fail the job, if the certificate cannot be loaded and is valid.

***SAME** - No changes to current settings.

***RQD REQUIRED** - Anytime the PGPPREC is evoked in the PKPGPZ command, the recipient defined here is required for the ZIP process to complete successfully.

***OPT OPTIONAL** - If the certificate cannot be loaded or is invalid the process will continue.

LDAPACTV

```
LDAP Definitions:
     Nbr Active LDAPs  . . . . . > 3            1-3, *NONE, *SAME
```

Or:

LDAPACTV(3)

LDAPACTV specifies the number of active LDAPs to be defined for Smartcrypt. The maximum is three.

LDAP1

```
  Primary LDAP:
     1 Network Name  . . . . . > '192.168.111.28'

     1 Port  . . . . . > 389           1-65535
     1 Sequence  . . . . . > 1             0-9
     1 TimeOut . . . . . > 0             0-9999
     1 Access User . . . . . > 'PKWAREADDEV\test'

     1 Access Passphrase . . . . > 'xxxx'

     1 Search Mode . . . . . > *EMAIL        *EMAIL, *CN
     1 Start Node  . . . . . > 'cn=users,dc=pkwareaddev,dc=com'

     1 Test  . . . . . > N             N, Y
```

Or:

LDAP1('192.168.111.28' 389 1 0 'PKWAREADDEV\test' 'xxxx' *EMAIL 'cn=users,dc=pkwareaddev,dc=com' N)

The LDAP1, LDAP2 and LDAP3 parameters define from 1 to 3 LDAP server configurations. To access the LDAP, see your directory services administrator for the LDAP servers. There are nine options for each LDAP.

Network Name (LDAP Server | *NONE |*SAME)

The LDAP network name can be either the TCP/IP address of the LDAP server or if name resolution is supported this can be the server's name. *NONE will blank out the network name.

Port (1-65353 |389)

Specifies the TCP port number the LDAP server is listening on. The standard LDAP port is 389, but each installation may vary, especially if it is a secured port.

Sequence (0-9)

Specifies the search sequence of databases and LDAP servers. This is reserved for future releases.

Timeout (0-9999)

Specifies the maximum number of seconds to wait for search results. 0 will use the default defined for the LDAP server.

Access User (Name of User)

The distinguished name of the entry user that will be used to bind.

Access Passphrase (Passphrase)

The credentials with which to authenticate the access user. Arbitrary credentials can be passed using this parameter. In most cases, this is the passphrase for the access user.

Search Mode (*CN|*EMAIL)

Specifies the default LDAP search mode that will be used if neither CN= nor EM= are prefixed on the recipient selection string.

Start Node (Distinguished name of where to start)

Specifies the distinguished name of the entry at which to start the search (at what portion of the LDAP structure should the search begin)

Test (Y|N)

Specify whether to Test the current LDAP. By coding this option to a Y for YES, then after the command has updated the security configuration file, a verification test will be performed for this LDAP. For more information on the results see the PKLDAPTST (Test LDAP) command.

LDAP Usage Notes:

Please be aware that the LDAP may not contain any encryption certificate validation policies. If the end user specifies only the LDAP, without a local CA and root stores for the environment, then the Smartcrypt default validation settings of TRUSTED and REVOKED will be enforced for the run. This will cause the run to fail during validation of the trusted certificate path because there are no CA and Root certificates available for processing. If you wish to execute the Smartcrypt with the LDAP only, then you will need to set your security environment policies or run policy for the authentication and encryption validation policies in the run for to exclude trust and revoke filters:

AUTHPOL(*WARN *NONE (*ALL *NOTTRUSTED *NOTREVOKED))
ENCRYPOL(*WARN (*ALL *NOTTRUSTED *NOTREVOKED))

NSSRULES

```
NSS Process Settings:
       NSS Required      . . . . . *SAME          *NO, *OPT, *REQ, *SAME
       NSS Classify Archive. . .   *SAME
               + for more values

       NSS Check Archive State .   *SAME          *NO, *WARN, *FAIL, *SAME
               + for more values
```

Or:

NSSRULES (*NO *SAME *SAME)
NSSRULES (*NO TOPSECRET_SUITEB_REQPLUS *WARN)

NSSRULES Secure Settings are the base security settings for NSS processing and currently consist of 3 options.

NSS Required (*NO | *OPT | *REQ |*SAME)

By setting this option to *OPT or *REQ activates the security configuration file. *NO indicates there are no requirements settings.

Options:

***NO**	No options are required.
***OPT**	This forces the use of NSS processing by either using the definitions of the main security settings or by using the parameters in the PKZIP or PKUNZIP commands .
***REQ**	*REQ forces the use of NSS processing by using the parameters defined in PKCFGSEC NSSRULES. i.e. Does not allow overrides by the PKZIP and PKUNZIP commands.
***SAME**	Current setting will not change.

NSS Classify Archive (*SYSTEM | *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 INACTIVE and, unless it is modified, Smartcrypt will use this enterprise setting.

Options:

***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.

***SAME** Current setting will not change.

NSS Check Archive State (*NO | *WARN | *FAIL |*SAME)

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.

Options:

***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.

***SAME** Current setting will not change.

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 versions of Windows prior to 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 security configuration parameter SECTYPE option "AES 3DES Keys" is introduced with ENTPREC processing to allow the Smartcrypt user to create AES-encrypted files that are compatible with older Windows workstations. 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 between 6.0 and 7.x due to Optimal Asymmetric Encryption Padding (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.

7 PKLDAPTST “LDAP Test” Command

Requires Smartcrypt

PKLDAPTST Command Summary with Parameter Keyword Format

PKLDAPTST is a utility command to assist in testing the LDAP servers that Smartcrypt will use. This command is included to assist with the LDAP parameters and to verify LDAP access along with time access.

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 command process reference the IBM Home page for your version of the operating system.

```
ACTION (    ({*SUMMARY}     )
             {*DETAIL }


SNAME (      Server Name or IP                    )
                {Network IP address or server Name }



PORT (      Port Number                    )
                              {389 }
                              {1-65535 }

USER (      Access User                    )
                             {LDAP User for Access }


PASSWORD (      Access Passphrase                 )
                            {User Passphrase }


STRNODE (      Start Node                 )
                            {Start Node Text String }
```

```
SEARCHFT (     Search Filter                      )
                           {'(cn=*)(userCertificate=*)'}
                           { valid LDAP Search String }


MAXI (     Max Item                       )
                             {0-9999 }
```

PKLDAPTST Command Keyword Details

ACTION - Action Type

ACTION (*SUMMARY|*DETAILS)

Specifies how much to display.

*SUMMARY will show the times and the count of items returned.

*DETAIL will show the common name of all LDAP items selected.

SNAME - Server Name or IP

SNAME (LDAP Server)

The LDAP Server name can be either the TCP IP address of the LDAP server or if name resolution is supported this can be the server's name.

To test an LDAP server defined to the system (with the PKCFGSEC command), enter

***LDAPn**

where n is the number (1 through 3) of the LDAP server. For example:

➔ **PKLDAPTST SNAME(*LDAP1)**

PORT - Port Number

PORT (389 | 1-9999)

Specifies the TCP port number the LDAP server is listening on. The standard LDAP ports is 389 but each installation may vary, especially if it is a secured port.

USER - Access User

USER (Distinguish name of an authorized user for access)

Specifies the distinguished name of the entry user that will be used to bind the LDAP server.

PASSWORD - Access Passphrase

PASSWORD (Passphrase of the Access User if available)

Specifies the credentials with which to authenticate the access user. Arbitrary credentials can be passed using this parameter. In most cases, this is the passphrase for the access user.

STRNODE - Start Node

STRNODE (Start Node Text String)

Specifies the distinguished name of the entry at which to start the search (at what portion of the LDAP structure should the search begin).

SEARCHFT - Search Filter

SEARCHFT ('(cn=*)(userCertificate=*)'| valid LDAP Search String)

Specifies the LDAP search string for the Test. The text must be a valid LDAP search string. The default string is '(CN=*)(userCertificate=*)', which will search for all common names that have a user certificate.

MAXI - Max Item

MAXI (100| 0-9999)

Specifies the maximum numbers of LDAP items to return for the test run. If this value exceeds the value defined for the LDAP server, it will take precedent over the MAXI value.

Sample Test:

```
                     Smartcrypt LDAP Test      (PKLDAPTST)

Type choices, press Enter.

Action Type  . . . . . . . . . .   *SUMMARY      *SUMMARY, *DETAIL
Server Name or IP  . . . . . . . > '192.168.132.25'
Port Number  . . . . . . . . . .   389           Character value
Access User  . . . . . . . . . . > 'PKWAREADDEV\test'
Access Passphrase  . . . . . . . .
Start Node . . . . . . . . . . . > 'cn=users,dc=pkwareaddev,dc=com'
```

```
Search Filter  . . . . . .   '(cn=*)(userCertificate=*)'
Max Item . . . . . > 100          Character value
```

Or:

PKLDAPTST SNAME('192.168.132.25') USER('PKWAREADDEV\test') PASSWORD() STRNODE('cn=users,dc=pkwareaddev,dc=com') MAXI(100)

Results:

```
   PKLDAPTEST LDAP Test Starting       2016/02/22 06:23:34

 PKLDAPTESTT Parameters:Action<T>
          - Server<192.168.132.25> Port<389>
          - User<PKWAREADDEV\test> Passphrase<6>
          - Start Node<cn=users,dc=pkwareaddev,dc=com>
          - Search Filter<(&(cn=*)(userCertificate=*))>
          - Max Items to Search<100> ,100
 LDAP_intialTest - --LDAP init ..... elasp time 0.000000 seconds
 LDAP_intialTest - --SizeLimit=100  TimeLimit=0
 LDAP_intialTest - --LDAP bind ..... elasp time 0.000000 seconds
 LDAP_intialTest - --LDAP Search ..... elasp time 0.000000 seconds
 LDAP_intialTest - --LDAP Search ...... 21 entries found
 LDAP_intialTest - --LDAP Attributes ..... elasp time 0.000000 seconds
 LDAP_intialTest - Total Entries=21
 PKLDAPTESTT LDAP Testing Ending RC=0
```

Or:

LDAP1('192.168.111.28' 389 1 0 'PKWAREADDEV\test' 'xxxx' *EMAIL'cn=users,dc=pkwareaddev,dc=com' N)

8 PKBLDCDB “Build Certificate Database” Command

Requires Smartcrypt

PKBLDCDB Command Summary with Parameter Keyword Format

PKBLDCDB is a utility command to build the certificate locator database files and set the security of the certificate file in the Integrated File System (IFS). When updating the database, PKBLDCDB checks the certificate to make sure it is trusted, not revoked, and not expired. If the certificate fails any of these checks, a warning message is issued, and information about the failed check is posted to the database for query purposes (see PKQRYCDB RUNTYPE(*SELECT)). To update the status of a certificate, re-process the certificate file with PKBLDCDB.

NOTE: In order to use PKBLDCDB, the user must have the proper authority for the path/file and for the CHGAUT command.

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.

```
MAINT (     Processing Type                       )
                           {*UPDATE }
                           {*TEST }
                           {*VERIFY }



MTYPE (     Maint. Type                  )
                           {*FILE }
                           {*DIRECTORY }


CTYPE (     Certificate Type                      )
                           {*PUBLIC }
                           {*PRIVATE }
```

```
FNAME (     Path/File Name              )
                          {Path and/or file name  }


PASSWORD (     Cert Passphrase                   )
                          {Certificate Private Key Passphrase }


DUPOPT (     Replace Duplicates                  )
                          {*REPLACE}
                          {*NOREPLACE}


LOGLVL (     Logging Level               )
                          {*LOG }
                          {*NOLOG }
                          {*MAXLOG }

OWNER (     Cert. Owner                  )
                          {*NONE }
                          {Valid User Id }



PUBAUTH (     PUBLIC Authority Settings:                 )
                 New data authorities   {*SAME }
                                                        {*RWX}
                                                        {*RX}
                                                        {*RW}
                                                        {*WX}
                                                        {*R }
                                                        {*W }
                                                        {*X }
                                                        {*EXCLUDE}
                  New object authorities   {*SAME }
                                                         {*ALL }
                                                         {*OBJEXIST}
                                                         {*OBJMGT}
                                                         {*OBJALTER}
                                                         {*OBJREF}
                                                         {*NONE}


UIDAUTH (     New ID Authority Settings:                 )
                     New User ID             {*CURRENT }
                                             {*NONE}

                                                        {Valid User Id }
                  New data authorities     {*SAME }
                                                        {*RWX}
                                                        {*RX}
                                                        {*RW}
                                                        {*WX}
                                                        {*R }
                                                        {*W }
                                                        {*X }
                                                        {*EXCLUDE}

                                      New object authorities   {*SAME }
                                                        {*ALL }
                                                        {*OBJEXIST}
                                                        {*OBJMGT}
                                                        {*OBJALTER}
                                                        {*OBJREF}
                                                        {*NONE}
```

PKBLDCDB Command Keyword Details

MAINT - Processing Type

MAINT (*UPDATE |*TEST |*VERIFY)

The processing type determines the action that PKBLDCDB.

*UPDATE will update the certificate locator database.

*VERIFY will scan the certificate locator database and validate that all the files in the database still exist in their store. If the DUPOPT parameter is *REPLACE and the file in the database does not exist, that database record will be removed.

*TEST will simulate updating the certificate locator database, but will not perform any posting. This command is helpful to review for errors and/or counts.

MTYPE - Maintenance Type

MTYPE (*FILE|*DIRECTORY)

The maintenance type determines the type of path/file name in the parameter FNAME. If *FILE, then FNAME should be a specific certificate file (fully qualified path included). If *DIRECTORY, the FNAME should be the fully qualified path that will be scanned for the certificate file.

CTYPE - Certificate Type

CTYPE (*PUBLIC|*PRIVATE)

CTYPE specifies the type of certificates (private or public) will be processed in this run.

*PUBLIC specifies that only a public certificate should be processed. No passphrase should be supplied.

*Private indicates that only private-key certificates should be processed and requires a passphrase be entered.

FNAME - Path/File Name

FNAME (Path and/or file name)

The contents of FNAME contain the IFS file or directory that will be used to update the certificate locator database. To use a specific file, code MTYPE(*FILE) and enter the full path and file name of the specific certificate file. If MTYPE(*DIRECTORY), then enter the IFS fully qualified directory that PKBLDCDB will scan for certificates.

If you have all your public certificates in one store, it is easy to use *DIRECTORY to update the database in one run. For the most part, the private key processing will

usually use *FILE for the run since private key certificates require a unique passphrase.

If you want to use the path to the system store for public or private key certificates, and you want to enter only the file name, prefix the file name with {SP}. This inserts the appropriate system path for the path of the file name.

For example: FNAME('{SP}myfile.cer'). If CTYPE(*PUBLIC), the path of the enterprise store will be something like '/myroot/cstore/public/myfile.cer'.

PASSWORD - Certificate Passphrase

PASSWORD (Certificate Private Key Passphrase)

To access the contents of a private key certificate, processing requires the passphrase assigned when the certificate was exported. When entering this passphrase, note that it is used only to open the certificate to gather the database data and is not stored or saved. The certificate is not altered in any way.

DUPOPT - Replace Duplicates

DUPOPT (*REPLACE|*NOREPLACE)

The DUPOPT is used with MAINT(*UPDATE) and MAINT(*VERIFY). When using MAINT(*UPDATE), DUPOPT(*REPLACE) will replace the database entry with new data when a duplicate entry is encountered. *NOREPLACE will only occur if a duplicate was not found.

When using MAINT(*VERIFY) and a file in the certificate locator database cannot be found, PKBLCDB will remove the database entry for that file with DUPOPT(*REPLACE). With *NOREPLACE only an error message will be displayed.

LOGLVL - Logging Level

LOGLVL (*LOG|*NOLOG |*MAXLOG)

Specifies the level of logging (printing/viewing) during a PKBLDCDB run. LOGLVL(*NOLOG) will only show a minimal amount of information. LOGLVL(*MAXLOG) will show more details, with some of the detail useful only for problem determination.

OWNER - Cert. Owner

OWNER (*NONE | New Object Owner ID)

OWNER transfers object ownership for the IFS certificate files processed during a PKBLDCDB run from the current owner to the specified owner. The authority that other users have to the object does not change. Using *NONE will make no ownership changes.

Note: In order for this change to work, the user running PKBLDCDB must have the authority to execute the CHGOWN command and must have the authority to change the files ownership. Otherwise an error will occur. The database will be updated but the ownership will not be changed.

PUBAUTH - PUBLIC Authority Settings

```
PUBLIC Authority Settings:
  New data authorities . . . . .   *SAME          *SAME, *RWX, *RX, *RW, *WX...
  New object authorities . . . .   *SAME          *SAME, *ALL, *OBJEXIST...
               + for more values
```

New data authorities {*SAME |*RWX |*RX |*RW |*WX |*R |*W |*X |*EXCLUDE}

New object authorities {*SAME | *ALL | *OBJEXIST | *OBJMGT | *OBJALTER | *OBJREF | *NONE}

The PUBAUTH allows the PKBLDCDB run to change the PUBLIC authorities on the IFS files that are processed. By specifying "PUBAUTH(*SAME (*SAME))", PKBLDCDB will not alter the file authority for PUBLIC.

The rules for the PUBAUTH parameter are the same as the CHGAUT command. For more information about changing PUBLIC authorities and the explanation of the data and object authorities, see the CHGAUT command.

Note: In order for this change work, the user running PKBLDCDB must have the authority to execute the CHGAUT command and must have the authority to change the IFS file authorities. Otherwise an error will occur. The database will be updated but the authorities will not be changed.

For details of the new data authorities and new object authorities, see the UIDAUTH parameter where the user would specify *PUBLIC.

UIDAUTH - New ID Authority Settings

```
New ID Authority Settings:
  New User ID  . . . . . .   *NONE          User Id, *CURRENT, *NONE
  New data authorities . . . . .   *SAME          *SAME, *NONE, *RWX, *RX...
  New object authorities . . . .   *SAME          *SAME, *NONE, *ALL...
               + for more values
```

New User ID {*NONE |*CURRENT | Valid User ID}

New data authorities {*SAME |*RWX |*RX |*RW |*WX |*R |*W |*X |*EXCLUDE}

New object authorities {*SAME | *ALL | *OBJEXIST | *OBJMGT | *OBJALTER | *OBJREF | *NONE}

The UIDAUTH allows the PKBLDCDB run to add/change a user's authorities on the IFS files that are processed. Specifying UIDAUTH(*NONE), no user authorities will be changed.

The rules for UIDAUTH parameter are the same as the CHGAUT command. For more information about changing object authorities and the explanation of the data and object authorities see the CHGAUT command.

Note: The user running PKBLDCDB must have the authority to execute the IBM i OS CHGAUT command and must have the authority to change the IFS file authorities. Otherwise an error will occur. The database will be updated but the authorities will not be changed.

UIDAUTH options are:

New User ID:

Specifies the user ID to whom authorities for the named file object are being given. If user ID is not *NONE, the authorities are given specifically to that user. The user ID must be a valid iSeries user ID. *CURRENT will use the ID of the user that is running the PKBLDCDB command.

New Data Authorities for the file:

Specifies the data authorities being given to the user specified in the new user ID parameter. If a value other than *SAME is specified, the value replaces any data authorities (*OBJOPR, *READ, *ADD, *UPD, *DLT, and *EXECUTE) that the user currently has to the objects.

The possible values are:

*SAME	The user's data authorities to the objects do not change.
*NONE	The user does not have any of the data authorities to the objects.
*RWX	The user is given *RWX authority to the objects. The user is given *RWX authority to perform all operations on the object except those limited to the owner or controlled by object existence, object management, object alter, and object reference authority. The user can change the object and perform basic functions on the object. *RWX authority provides object operational authority and all the data authorities.
*RX	The user is given *RX authority to perform basic operations on the object, such as run a program or display the contents of a file. The user is prevented from changing the object. *RX authority provides object operational authority and read and execute authorities.
*RW	The user is given *RW authority to view the contents of an object and change the contents of an object. *RW authority provides object operational authority and data read, add, update, and delete authorities.
*WX	The user is given *WX authority to change the contents of an object and run a program or search a library or directory. *WX authority provides object operational authority and data add, update, delete, and execute authorities.

*R	The user is given *R authority to view the contents of an object. *R authority provides object operational authority and data read authority.
*W	The user is given *W authority to change the contents of an object. *W authority provides object operational authority and data add, update, and delete authorities.
*X	The user is given *X authority to run a program or search a library or directory. *X authority provides object operational authority and data execute authority.
*EXCLUDE	Exclude authority prevents the user from accessing the object.

New Object Authorities for the file(s):

Specifies the object authorities being given to the user specified in the user parameter. If a value other than *SAME is specified, the value replaces any object authorities (*OBJEXIST, *OBJMGT, *OBJALTER, and *OBJREF) that the user currently has to the objects.

The possible values are:

*SAME	The user's object authorities to the objects do not change.
*NONE	The user does not have any other object authorities (existence, management, alter, or reference). If *EXCLUDE or *AUTL is specified for the DTAAUT parameter, this value must be specified.
*ALL	All of the other object authorities (existence, management, alter, and reference) are given to the users. Or specify up to four (4) of the following values:
*OBJEXIST	The user is given object existence authority to the object.
*OBJMGT	The user is given object management authority to the object.
*OBJALTER	The user is given object alter authority to the object.
*OBJREF	The user is given object reference authority to the object.

9 PKQRYCDB “Query Cert Database” Command

Requires Smartcrypt

PKQRYCDB Command Summary with Parameter Keyword Format

PKQRYCDB is a utility command to query the certificate locator database files, a certificate file in the IFS, or an OpenPGP keyring file in the 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           *DB, *FILE, *P7B, *PLINKA...
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| *PGPKRF)

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, the query will display the details for the contents of the CRL file in FNAME.

*PKCS12, the query will display the details for the contents of the PKCS12 file in FNAME. A password is required to open the PKCS12 package.

If *PGPKRF, the query will display the details for the contents of an OpenPGP keyring file in FNAME. The keyring file must be in an IFS path.

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 of 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------2016/02/16 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------2016/02/16 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------2010/04/16 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 2009
NotAfter:
   Wed Jul 21 18:59:59 2013
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
SerialNumber	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 key ring 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.

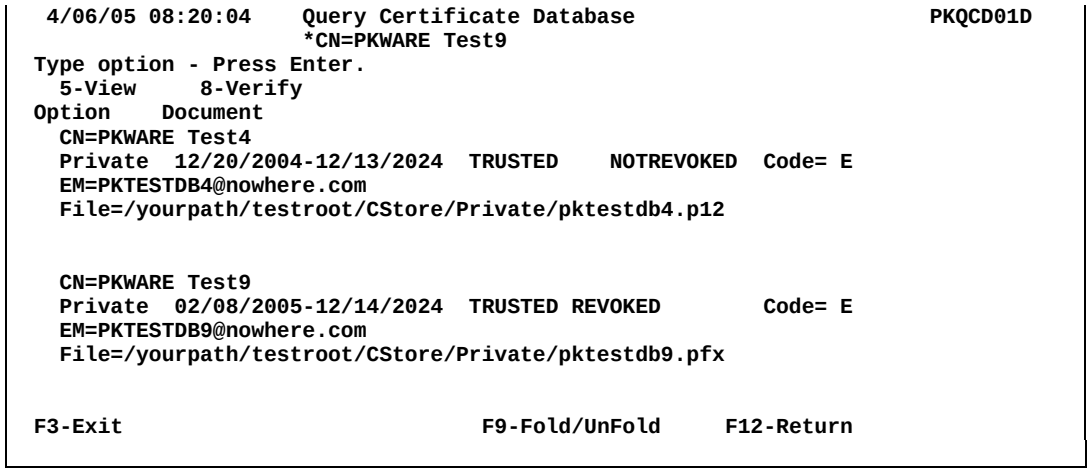
➔ **PKQRYCDB RUNTYPE(*SELECT) FNAME('cn=P*')**

Folded

```
  4/06/05 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

```
  4/06/05 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


  4/06/05 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
```


4/06/05 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

10 PKSTORADM “Certificate Store Administration” Command

Requires Smartcrypt

PKSTORADM Command Summary with Parameter Keyword Format

PKSTORADM is a certificate store administration utility command to maintain the certificate authority (CA), trusted root, and certificate revocation list (CRL) stores. Before using this utility, define the three stores and their location with PKCFGSEC command.

X.509 certificates may be imported to the local certificate store through the Smartcrypt local certificate store administration tool PKSTORADM. These certificates are frequently obtained through another platform and (binary) transferred to the operational iSeries system for installation.

Important Note: All X.509 certificates should be transferred to the local iSeries environment in binary mode with no translation.

When certificates are imported, you may either define which store a certificate should be placed in or let the certificate administration tool determine the appropriate store location based on the certificate type. You must tell the PKSTORADM what certificate file type and key type to import.

iSeries UPDATE authority (or equivalent) must be granted to the system administrator responsible for altering the certificate store.

PKSTORADM can import certificates to be added to your CA and/or to your TRUSTED ROOT STORES. It can also be used to import a CRL into your static CRL store. When building your stores, you should first install the trusted roots, followed by the CA certificates. Before importing a CRL file, its CA must be in the CA store.

Usage Notes

If a certificate is designated to be an end-entity certificate, it can be ignored, or a `.cer` file will be created in the public store path.

It’s important that only known trusted roots be allowed to be added to your trusted root store. You should install a certificate to your root store only when you have

confirmed its authenticity. To do this, contact the certificate authority listed in the candidate root certificate file. If you install a certificate to your root without confirming its authenticity, you may be creating a security risk. Once you install the certificate to your root store, Smartcrypt will use it to complete future certificate trust chain validation processing associated with the certificate authority.

PKSTORADM uses the Smartcrypt utility PKCAADMN and most of the message issued will be from that utility. Messages that start with prefix ZPCA are from the PKCAADMN utility and be found in the message section. If there are no error messages, AQZ0049 will be issued and can be monitored. If an error is encountered, the escape message AQZ0050 will be issued and can be monitored.

Keywords are demarcated by spaces. In many cases there are multiple entries for a parameter, with each entry again demarcated by spaces. For more information about the command process, refer to the IBM home page for your version of the operating system.

```
                  Smartcrypt CA Store Admin   (PKSTORADM)

Type choices, press Enter.

Processing Type  . . . . . .  _________     *IMPORT, *BROWSE, *LIST...
Preferred Store  . . . . . .                *CA, *ROOT, *CRL, *DETECT...
File Type  . . . . . .  . . .  *CER         *CER, *P7B, *CRL, *PKCS12
End Entities:
  Process End Entities?  . . . .  *NO       *YES, *NO
  File Mode  . . . . . .  . . .  *UNIQUE    *UNIQUE, *OVERWRITE
  Update Database  . . . . . .  *YES        *YES, *NO
Selection Name . . . . . .  ____________________________________________

Cert Passphrase. . . . . .  ____________________________________________
Logging Level  . . . . . .  *LOG          *NOLOG, *LOG, *MAXLOG
Temporary Controls . . . . . .  *NONE      *SIMULATE, *NONE, *CHKVER
Certificate Controls  . . . . . .  *STOP   *STOP, *IGNORE, *ENDENTITY
```

PKSTORADM Command Keyword Details

RUNTYPE - Processing Type

RUNTYPE (*IMPORT |* LIST |*BROWSE |*VERIFY |*DELETE |*INITIALIZE)

Specifies the processing type or action that PKSTORADM will perform:

***IMPORT**	Reads the input file (defined in the FNAME) with the file type (FTYPE) and add the certificates to the appropriate store defined with the preferred store STYPE.
***LIST**	Provides a complete, detailed listing of all the certificates in the store defined in the STYPE parameter. No updating takes place.
***BROWSE**	Provides a short display of all the certificates in the store defined in the STYPE parameter. No updating takes place.

***VERIFY**	Tests the store defined with the STYPE parameter and verifies the condition of your store. No updating takes place.
***DELETE**	Delete a certificate from either the CA or root store. First a browse list similar to the browse list for the short list will be displayed. Enter a '4' for the option of the certificate to be deleted and press the ENTER key to display a confirmation message window. To complete the deletion, press F5 to delete or F12 to cancel. If *DETECT or *ALL is entered both stores will be displayed. Only one delete option of '4' can be entered.
***INITIALIZE**	Clears and initializes the store defined in the STYPE parameter. For this command to work, you must also type the word 'Initialize' in the FNAME parameter. **Note**: Take care when using the option because it clears all of your store's current content.

FTYPE - File Type

FTYPE (*CER | *P7B | *CRL | *PKCS12)

Defines the format or package type that the input file contains.

***CER (PEM)**	Contains a single public-key certificate. It may be in Base-64 encoded (ASCII text with ASCII headers) or DER-encoded binary format. Common file extensions: `.pem, .cer, .key`
***P7B (PKCS#7)**	Contains one or more CA (and or root) certificates or could also contain one or more CRL in a PKCS#7 format. Common file extension: `.p7b`
***CRL**	Contains certificate revocation list in a CRL format issued by a certificate authority. Common file extension: `.crl`
*****PKCS12	Contains one or more certificates with private keys. Password is required. Common file extension: `.pfx, .p12`

STYPE - Certificate Store

STYPE (*DETECT | *ALL | *CA |*ROOT |*CRL)

STYPE specifies the type of certificates, private or public, to be processed in this run.

***DETECT**	Detect will try to determine which store the certificate will be imported into. This is also known as Best Guess (BG) option. If *DETECT is used with RUNTYPE(*LIST) or RUNTYPE(*VERIFY), it is assumed to perform on all stores.

***ALL**	The same as using *DETECT.
***CA**	Certificate authority store defined in the CSCA parameter of PKCFGSEC.
***ROOT**	Trusted root store defined in the CSROOT parameter of PKCFGSEC.
***CRL**	Static CRL (Certificate Revocation List) store defined in the CSCRL parameter of PKCFGSEC.

ENDENTITY - End Entities

```
End Entities:
  Process End Entities?  . . . .   *NO           *YES, *NO
  File Mode  . . . . . . . . . .   *UNIQUE       *UNIQUE, *OVERWRITE
  Update Database  . . . . . . .   *YES          *YES, *NO
```

Or:

ENDENTITY(*YES *OVERWRITE *YES)

ENDENTITY specifies whether to automatically update end entity certificates when found through the PKBLDCDB process by building X.509 files in a DER-encoded binary format and placing the files in the public store path. The file name will be the name of the inputted file with a _EEx.cer suffixed at the end, where x is a sequence number. After adding you should run PKBLDCDB to record the latest public certificates.

Process End Entities? (*YES|*NO)

Specifies whether to build the end entity certificates in the public store path.

***YES**	If a p7b type file is being read that contains end entity certificates, the end entity certificates will have a file created for each certificate in the public store path, and a PKBLDCDB will be launched to update the database.
***NO**	Ignore end entity certificates.

File Mode? (*UNIQUE|*OVERWRITE)

Specifies action to take if a duplicate file is found in the public store path.

***UNIQUE**	If a file is being generated and a duplicate name is found in the public store path, then the number will be incremented until no duplicate is found.
***OVERWRITE**	If a duplicate file is found in the public store path, the file will be written over with the new file. If a duplicate is written over, then you should run a PKBLDCDB with *VERIFY to correct the locator database.

Update Database (*YES |*NO)

Specifies that when an end entity certificate file is created in the public path, that the PKBLDCDB command will be issued for each file.

***YES**	PKBLDCDB will be issued for all end entity certificate files that are created in the run.
***NO**	The PKBLDCDB will not be issued for the end entity certificate files that are created.

FNAME - File Name

FNAME (Path/File name)

FNAME is the inputted IFS path and file name that will be used to import with the RUNTYPE(*IMPORT). If you specify RUNTYPE(INITIALIZE), then you must enter 'Initialize' to initialize a store.

For RUNTYPE values *LIST and *BROWSE, FNAME can also be used for a string search field. If the FNAME starts with an *, the contents after the asterisk are used for a string search on the certificate's friendly name. For example, if FNAME is *PKWARE, PKSTORADM returns only certificates that contain the string PKWARE in the friendly name. The search is not case sensitive.

PASSWORD - Certificate Passphrase

PASSWORD (Certificate Private Key Passphrase)

Reserved for future use. To process passphrase protected P7B files that may contain a private key.

LOGLVL - Logging Level

LOGLVL (*LOG|*NOLOG |*MAXLOG)

Specifies the level of logging (printing/viewing) used during a PKSTORADM run. LOGLVL(*NOLOG) shows a minimal amount of information. LOGLVL(*MAXLOG) shows more details, with some of detail useful only for problem determination.

CNTRLS - Temporary Controls

CNTRLS (*NONE |*SIMULATE | *CHKVER)

Specifies special process controls for the run. At this time only *SIMULATE exits.

***NONE**	No special controls.
***SIMULATE**	Only valid with the RTYPE(*IMPORT). Simulates the run process and does *not* update the stores. Provides a way to see how and what would be added if an input file is added to the stores.
***CHKVER**	Reserved for Future Use. Planned to verify the signing hash used in the certificates that they are acceptable to

Smartcrypt. Accepted signing hashes are SHA1 and MD5.

CERTFLG- Certificate Controls

CERTFLG (*STOP, *IGNORE, *ENDENTITY)

Defines the action to take when the certificates type (CA, ROOT, or End Entity) cannot be determined utilizing the basic constraints.

***STOP**	Stop and indicate an error without updating any of the stores.
***IGNORE**	Bypass the unknown type certificate and continue with other certificates if any.
***ENDENTITY**	If the type cannot be determined default the certificate's type to be an End Entity certificate.

Sample Displays

The following example shows a simulation of adding an inputted CER type file to the root store.

➔ **PKSTORADM RUNTYPE(*IMPORT) FTYPE(*CER) STYPE(*ROOT) FNAME('/myroot/pkware/CStore/Testzips/pkwareRoot.cer') CNTRLS(*SIMULATE)**

```
PKSTORADM Smartcrypt CA Store Administration starting------2005/03/14 13:35:57
ZPCA000I SUCCESS: Added certificate to store
'/myroot/pkware/CStore/Root/pkwareRT.store'.
DSN=/myroot/pkware/CStore/Testzips/pkwareRoot.cer;CER=1;FN=PKTESTDB
Root;TP=A91CD312EFFEF51C8ABFEFD92C23FF67FBDBEBE3;SN=00;E=PKTESTDBRoot@nowhere.com
;ROOT
ZPCA846W WARNING: Simulation Requested.  Nothing will be saved to the store.

ZPCA000I SUCCESS: Saved certificate store
'/myroot/pkware/CStore/Root/pkwareRT.store' to disk.

ZPCA000I Added 0 of a possible 1 certificates to the ROOT store.
ZPCA000I 0 certificates in the ROOT store before the Add command.
ZPCA000I 0 certificates in the ROOT store after the Add command.
ZPCA846W WARNING: Simulation Requested.  Nothing will be saved to the store.

PKSTORADM Store Admin ending------0
PKSTORADM  Completed Successfully
```

The following example shows adding an inputted CER type file to the root store. Since a CER type file only has one certificate, one of a possible one certificate(s) was added. Since the store was initially empty, the number of certificates in the store before adding was zero, and the number after the add is one.

➔ **PKSTORADM RUNTYPE(*IMPORT) FTYPE(*CER) STYPE(*ROOT) FNAME('/myroot/pkware/CStore/Testzips/pkwareRoot.cer')**

```
PKSTORADM Smartcrypt CA Store Administration starting------2005/03/14 13:47:55
ZPCA000I SUCCESS: Added certificate to store '/myroot/pkware/CStore/Root/
```

```
pkwareRT.store'.
DSN=/myroot/pkware/CStore/Testzips/pkwareRoot.cer;CER=1;FN=PKTESTDB
Root;TP=A91CD312EFFEF51C8ABFEFD92C23FF67FBDBEBE3;SN=00;E=PKTESTDBRoot@nowhere.com
;ROOT

ZPCA000I SUCCESS: Saved certificate store
'/myroot/pkware/CStore/Root/pkwareRT.store' to disk.

ZPCA000I Added 1 of a possible 1 certificates to the ROOT store.
ZPCA000I 0 certificates in the ROOT store before the Add command.
ZPCA000I 1 certificates in the ROOT store after the Add command.


PKSTORADM Store Admin ending------0
PKSTORADM  Completed Successfully
```

This example also adds a certificate, but this time the file is destined for the CA store.

➔ **PKSTORADM RUNTYPE(*IMPORT) FTYPE(*CER) STYPE(*CA) FNAME('/myroot/pkware/CStore/Testzips/pkwareCA.cer')**

```
PKSTORADM Smartcrypt CA Store Administration starting------2005/03/14 13:51:22
ZPCA000I SUCCESS: Added certificate to store
'/myroot/pkware/CStore/CA/pkwareCA.store'.
DSN=/myroot/pkware/CStore/Testzips/pkwareCA.cer;CER=1;FN=PKWARE Test Intermediate
Cert;TP=2B3C27C30067690EFB77A8A84DAB1D39A1368906;SN=01;E=PKTESTDBIC@nowhere.com;C
A

ZPCA000I SUCCESS: Saved certificate store
'/myroot/pkware/CStore/CA/pkwareCA.store' to disk.

ZPCA000I Added 1 of a possible 1 certificates to the CA store.
ZPCA000I 0 certificates in the CA store before the Add command.
ZPCA000I 1 certificates in the CA store after the Add command.



PKSTORADM Store Admin ending------0
```

This example shows the verification action. Since the store type is STYPE(*DETECT), it verifies all of the stores.

➔ **PKSTORADM RUNTYPE(*VERIFY) FTYPE(*CER) STYPE(*DETECT)**

```
PKSTORADM Smartcrypt CA Store Administration starting------2005/03/14 13:53:41
ZPCA000I SUCCESS: CA Store '/myroot/pkware/CStore/CA/pkwareCA.store' verified
successfully.  1 certificates found.

ZPCA000I SUCCESS: Root Store '/myroot/pkware/CStore/Root/pkwareRT.store' verified
successfully.  1 certificates found.

ZPCA000I SUCCESS: CRL store '/myroot/pkware/CStore/CRL/pkwareCRL.store'
successfully verified.  0 revocation lists found.



PKSTORADM Store Admin ending------0
```

The following example does a detail list. Since the store type is STYPE(*DETECT), all of the stores are listed. The list can get quite lengthy but may be needed to verify your store contents.

➔ **PKSTORADM RUNTYPE(*LIST) FTYPE(*CER) STYPE(*DETECT)**

```
PKSTORADM Smartcrypt CA Store Administration starting------2005/03/14 13:57:38

 CA Store
 --------

 --- Certificate 1 ---
 PKWARE Test Intermediate Cert
 Subject:
    C=US
    S=Wisconsin
    L=Milwaukee
    O=PKWARE, Inc.
    OU=PKWARE, Inc. -- for test and evaluation purposes only
    CN=PKWARE Test Intermediate Cert
    E=PKTESTDBIC@nowhere.com
 Issuer:
    C=US
    S=Wisconsin
    L=Milwaukee
    O=PKWARE, Inc.
    OU=PKWARE, Inc. -- for test and evaluation purposes only
    CN=PKTESTDB Root
    E=PKTESTDBRoot@nowhere.com
 SerialNumber:
    01
 NotBefore:
    Mon Dec 20 08:54:09 2004
 NotAfter:
    Sat Dec 14 08:54:09 2024
 SHA-1 Hash of Certificate(Thumbprint):
    2B 3C 27 C3 00 67 69 0E FB 77 A8 A8 4D AB 1D 39 A1 36 89 06
 Public Key Hash:
    72 C0 6D 05 C9 3E A9 6C 34 9C AD D0 8F 14 C0 8E 04 B0 E5 CF
 Certificate Authority

 --- Certificate 2 ---
PKWARE Test Intermediate Cert A
Subject:
   C=US
   S=Wisconsin
   L=Milwaukee
   O=PKWARE, Inc.
   OU=PKWARE, Inc. -- for test and evaluation purposes only
   CN=PKWARE Test Intermediate Cert A
   E=PKTESTDBICA@nowhere.com
Issuer:
   C=US
   S=Wisconsin
   L=Milwaukee
   O=PKWARE, Inc.
   OU=PKWARE, Inc. -- for test and evaluation purposes only
   CN=PKTESTDB Root
   E=PKTESTDBRoot@nowhere.com
SerialNumber:
   02
NotBefore:
   Tue Feb  8 11:38:17 2005
NotAfter:
   Sun Dec 15 11:38:17 2024
SHA-1 Hash of Certificate(Thumbprint):
   C4 05 A5 BC 36 94 21 49 D5 C2 12 CB 6C 21 3C 4F 1F E8 93 E6
Public Key Hash:
   08 3B E9 79 78 15 E7 BA E9 00 90 00 D5 AC 92 BD 83 7A 8D 57
Certificate Authority



Root Store
```

```
----------

--- Certificate 1 ---
PKTESTDB Root
Subject:
   C=US
   S=Wisconsin
   L=Milwaukee
   O=PKWARE, Inc.
   OU=PKWARE, Inc. -- for test and evaluation purposes only
   CN=PKTESTDB Root
   E=PKTESTDBRoot@nowhere.com
Issuer:
   C=US
   S=Wisconsin
   L=Milwaukee
   O=PKWARE, Inc.
   OU=PKWARE, Inc. -- for test and evaluation purposes only
   CN=PKTESTDB Root
   E=PKTESTDBRoot@nowhere.com
SerialNumber:
   00
 NotBefore:
   Mon Dec 20 08:21:34 2004
NotAfter:
   Thu Dec 19 08:21:34 2024
SHA-1 Hash of Certificate(Thumbprint):
   A9 1C D3 12 EF FE F5 1C 8A BF EF D9 2C 23 FF 67 FB DB EB E3
Public Key Hash:
   9A C2 BA 79 76 20 64 5E 12 79 D9 74 4C A8 59 66 71 E9 C2 DD
Self Signed
Certificate Authority



CRL Store
---------


PKSTORADM Store Admin ending------0
```

The following example adds a CRL to the CRL store.

➔ **PKSTORADM RUNTYPE(*IMPORT) FTYPE(*CRL) STYPE(*CRL) FNAME('/myroot/pkware/CStore/Testzips/crl1.crl')**

```
PKSTORADM Smartcrypt CA Store Administration starting------2005/03/14 14:16:48
ZPCA000I SUCCESS: Added certificate '/myroot/pkware/CStore/Testzips/crl1. crl' to
store '/myroot/pkware/CStore/CRL/pkwareCRL.store'.

ZPCA000I SUCCESS: Saved certificate store '/myroot/pkware/CStore/CRL/pkwa
reCRL.store' to disk.

ZPCA000I Added 1 out of 1 certificates to the CRL store.
ZPCA000I 0 entries in the CRL store before the Add command.
ZPCA000I 1 entries in the CRL store after the Add command.



PKSTORADM Store Admin ending------0
PKSTORADM  Completed Successfully
```

The following example displays a CRL listing.

➔ **PKSTORADM RUNTYPE(*LIST) STYPE(*CRL)**

```
PKSTORADM Smartcrypt CA Store Administration starting------2005/03/14 14:20:59

CRL Store
---------
--- CRL 1 ---
   PKWARE Test Intermediate Cert A
Issuer:
   C=US;S=Wisconsin;L=Milwaukee;O=PKWARE, Inc.;OU=PKWARE, Inc. -- for test and
evaluation purposes only;CN=PKWARE Test Intermediate Cert
A;E=PKTESTDBICA@nowhere.com
LastUpdate:
   Unknown
NextUpdate:
   Unknown
Revoked Serial Numbers (1):
SerialNumber=01;
IDHash=DA9F053EEF6684FC2BDF63962E24775EE81160ED;
PKSTORADM Store Admin ending------0
PKSTORADM  Completed Successfully
```

The following example demonstrates a *BROWSE display browse.

➔ **PKSTORADM RUNTYPE(*BROWSE) STYPE(*ALL)**

```
 3/23/05 08:17:01    Certificate Store Admin                          PKQCD01D
                         Store Review Query


 Option    Document
   CA    1    FN=Class 1 Public Primary Certification Authority
   CA    2    FN=Open Financial Exchange CA - Class 3,www.verisign.com/CPS Incor
   CA    3    FN=Open Financial Exchange CA - Class 3,www.verisign.com/CPS Incor
   CA    4    FN=PKWARE Test Intermediate Cert E
   CA    5    FN=VeriSign Class 2 CA - Individual Subscriber
   CA    6    FN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
   CA    7    FN=Thawte Server CA
   CA    8    FN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
   CA    9    FN=Thawte Premium Server CA
   CA    10   FN=PKWARE Test Intermediate Cert
   CA    11   FN=PKWARE Test Intermediate Cert A
   CA    12   FN=PKWARE Test Intermediate Cert F
   CA    13   FN=VeriSign, Inc.,VeriSign International Server CA - Class 3,www.v
   Root  1    FN=VeriSign Commercial Software Publishers CA
   Root  2    FN=VeriSign Individual Software Publishers CA                  +


 F3-Exit                           F9-Fold     F12-Return
```

F9 to Unfold:

```
3/23/05 08:17:01    Certificate Store Admin                          PKQCD01D
                        Store Review Query


Option    Document
  CA    1    FN=Class 1 Public Primary Certification Authority


             SN=00CDBA7F56F0DFE4BC54FE22ACB372AA55

  CA    2    FN=Open Financial Exchange CA - Class 3,www.verisign.com/CPS Incor
. By Ref.,LIAB. LTD. (c) 97 VeriSign

             SN=1E47DBE0434AFB7AE84ABEF2EC2C7A6F

  CA    3    FN=Open Financial Exchange CA - Class 3,www.verisign.com/CPS Incor
```

```
. By Ref.,LIAB. LTD. (c) 97 VeriSign

             SN=5C1CD1200795F820DE576AC1FCCBF8C6410E0812
                                                                          +


F3-Exit                            F9-Fold     F12-Return
```

The following example demonstrates the importing of certificates from a p7b file that has the end entity certificate with the full path of CA and root certificates. If the CA and root already exist, the number of certificates in the stores will not increase.

Note the file created in the public store for the end entity certificate. See the message (*End Entity File </yourpath/PKWARE/Cstores/public/carolineF_EE1.cer> created*) where the suffix _EEx.cer was attached and the file written in the public store path. PKBLDCDB should now be run to add the certificate to the locator database. You should run with CNTRLS(*SIMULATE) first to make sure what you are adding to your CA and root stores is acceptable and trusted.

➔ **PKSTORADM RUNTYPE(*IMPORT) FTYPE(*P7B) STYPE(*DETECT) ENDENTITY(*YES *UNIQUE) FNAME('/yourpath/PKWARE/Cstores/Inputs/carolineF.p7b')**

```
PKSTORADM Smartcrypt CA Store Administration starting------2005/03/23 09:11:14

ZPCA000I SUCCESS: Added certificate to store
'/yourpath/PKWARE/Cstores/Root/pkwareRT.store'.
DSN=/yourpath/PKWARE/Cstores/Inputs/carolineF.p7b;CER=2;FN=Class 1 Public Primary
Certification Authority;TP=90AEA26985FF14804C434952ECE9608477AF556F
;SN=00CDBA7F56F0DFE4BC54FE22ACB372AA55;ROOT
ZPCA000I SUCCESS: Saved certificate store
'/yourpath/PKWARE/Cstores/Root/pkwareRT.store' to disk.
ZPCA000I SUCCESS: Added certificate to store
'/yourpath/PKWARE/Cstores/CA/pkwareCA.store '.
DSN=/yourpath/PKWARE/Cstores/Inputs/carolineF.p7b;CER=3;FN=VeriSign Class 1 CA I
ndividual Subscriber-Persona Not Validated;TP=12519AE9CD777A560184F1FBD542152
22E95E71F;SN=0D8B4FEEAAD2185BF4756A9D29E17FFB;CA

ZPCA000I SUCCESS: Saved certificate store
'/yourpath/PKWARE/Cstores/CA/pkwareCA.store' t o disk.

ZPCA000I Added 1 of a possible 3 certificates to the stores.
ZPCA000I Added 0 certificates to the CA store.
ZPCA000I 12 certificates in the CA store before the Add command.
ZPCA000I 12 certificates in the CA store after the Add command.

ZPCA000I Added 0 certificates to the ROOT store.
ZPCA000I 27 certificates in the ROOT store before the Add command.
ZPCA000I 27 certificates in the ROOT store after the Add command.

ZPCA000I 1 end entities written to the
'/yourpath/PKWARE/Cstores/CWrkPath/SZdbfbecc7.tmp' output file.

End Entity File </yourpath/PKWARE/Cstores/public/carolineF_EE1.cer> created.


PKSTORADM Store Admin ending------0
PKSTORADM  Completed Successfully
```

Now add the public certificate to the locator database with the PKBLDCDB command:

➔ **PKBLDCDB MAINT(*UPDATE) MTYPE(*FILE) CTYPE(*PUBLIC) FNAME('/yourpath/PKWARE/Cstores/public/carolineF_EE1.cer')**

```
PKBLDCDB Update Smartcrypt Cert DataBase starting------2005/03/23 09:32:25
PKBLDCDB Running Update Mode--Replace Duplicates---
PKBLDCDB Adding  </yourpath/PKWARE/Cstores/public/carolineF_EE1.cer>
PKBLDCDB Run Totals:
   Total Records Added     =1
   Total Records Updated   =0
   Total Duplicates Found      =0
   Total Records In Error  =0
   Total Records Processed =1
PKBLDCDB Scan ending------
PKBLDCDB   Completed Successfully
```

The following example demonstrates the certificate deletion process:

➔ **PKSTORADM RUNTYPE(*DELETE) STYPE(*CA)**

```
 3/23/05 08:23:02    Certificate Store Admin                          PKQCD01D
                        Store Review Query
Type option - Press Enter.  ONLY 1 Delete Selection Valid
  4-Delete
Option    Document
  CA    1    FN=Class 1 Public Primary Certification Authority
  CA    2    FN=Open Financial Exchange CA - Class 3,www.verisign.com/CPS Incor
  CA    3    FN=Open Financial Exchange CA - Class 3,www.verisign.com/CPS Incor
  CA    4    FN=PKWARE Test Intermediate Cert E
  CA    5    FN=VeriSign Class 2 CA - Individual Subscriber
  CA    6    FN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
  CA    7    FN=Thawte Server CA
  CA    8    FN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
  CA    9    FN=Thawte Premium Server CA
  CA    10   FN=PKWARE Test Intermediate Cert
  CA    11   FN=PKWARE Test Intermediate Cert A
4 CA    12   FN=PKWARE Test Intermediate Cert F
  CA    13   FN=VeriSign, Inc.,VeriSign International Server CA - Class 3,www.v
                                                                             +

F3-Exit                          F9-Fold     F12-Return
```

Enter 4 on the certificate line to delete, and press the ENTER key. This will cause the following window to appear for confirmation.

```
 3/23/05 08:23:02    Certificate Store Admin                          PKQCD01D
                        Store Review Query
Type option - Press Enter.  ONLY 1 Delete Selection Valid
  4-D ..............................................................
Optio :  Confirm Certificate to be deleted from CA Store            :
  CA  :    Location: 12                                             :
  CA  :    FN= PKWARE Test Intermediate Cert F                      : or
  CA  :    SN= 01                                                   : or
  CA  :                                                             :
  CA  : Note:                                                       :
  CA  :   Certificates that are issued by the certification        : ed
  CA  :   authorities or any lower level certification authorities  :
  CA  :   will no longer be trusted.  Press  PF5 to Continue with   : ed
  CA  :   the deletions or PF12 to exit without deleting the        :
  CA  :   certificate.                                              :
  CA  :          Press F5 to Delete Certificate                     :
4 CA  :          Press PF12 to exit Without Delete                  :
  CA  :  F5=Delete  F12=Cancel                                      : .v
      :                                                             :
      :.............................................................: +
```

To delete the certificate, press the PF5 key.

```
PKSTORADM Smartcrypt CA Store Administration starting------2005/03/23 08:30:18

ZPCA000I SUCCESS: Deleted certificate from store.
ZPCA000I SUCCESS: Saved certificate store
'/yourpath/PKWARE/Cstores/CA/pkwareCA.store' to disk.
ZPCA000I Deleted 1 certificates from the CA store.
ZPCA000I 13 certificates in the CA store before the Delete command.
ZPCA000I 12 certificates in the CA store after the Delete command.
Certificate Deleted From CA Store


PKSTORADM Store Admin ending------0
PKSTORADM  Completed Successfully
```

11 PKWARE PartnerLink: SecureZIP Partner

This chapter applies only to participants in the PKWARE SecureZIP Partner program. Other readers may skip this section.

This chapter addresses administration activities unique to the ***SecureZIP Partner for IBM i*** product.

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 User's Guide* for an operational description of the SecureZIP Partner product.

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.

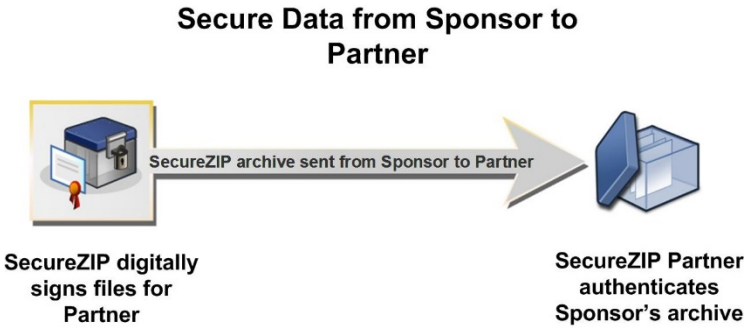
A SecureZIP Partner *sponsor* organization establishes a SecureZIP Partner relationship with another organization. As a SecureZIP Partner, the external organization receives the ***SecureZIP Partner for IBM i*** application to use to

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 PartnerLink extracts archives only *from a sponsor* and creates and encrypts archives only *for a sponsor*.

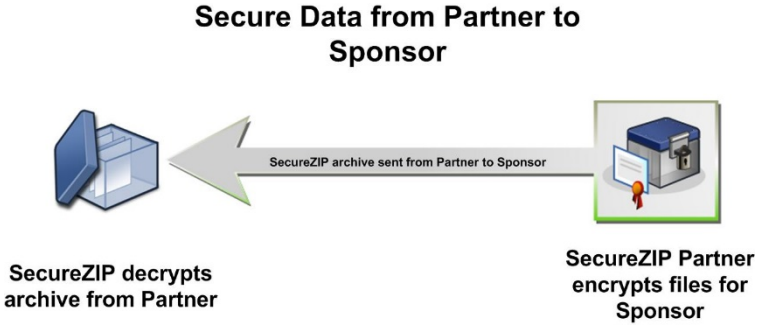
Decrypting and Extracting Sponsor Data (Read Mode)

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.



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.



Getting Started

Smartcrypt customers join the SecureZIP Partner program by contacting PKWARE and applying for a SecureZIP Partner sponsorship.

A SecureZIP Partner sponsor provides PKWARE with a copy of the public key matching the certificate that will be used to sign secure containers sent to partners. This key enables a partner to authenticate sponsor signatures.

A sponsor may also provide a copy of a public key for the partner to use to encrypt data files for the sponsor, and also a copy of a designated (public) contingency key. These encryption keys are needed only if a sponsor wants to enable partners to create archives for delivery to the sponsor. SecureZIP Partner creates only archives encrypted for a designated sponsor, using sponsor-provided keys. If a sponsor does not provide keys for encryption, a partner cannot use SecureZIP Partner to create archives. SecureZIP Partner does not create unencrypted archives.

PKWARE incorporates the sponsor keys into a SecureZIP Partner Sponsor Distribution Package (SDP). The Sponsor Distribution Package is used to configure a ***SecureZIP Partner*** installation to extract Smartcrypt secure containers signed by a sponsor and (if encryption keys are provided) to encrypt data files for a sponsor using the sponsor's public keys. ***SecureZIP Partner*** extracts archives only if they are signed by a sponsor. If keys for encryption are included in the SDP, ***SecureZIP Partner*** automatically encrypts archives created for the respective sponsor using the included keys. Only the sponsor recipients who own those keys can decrypt and read the files in an archive that ***SecureZIP Partner*** encrypts.

Once the Sponsor Distribution Package has been created, a sponsor can invite outside partner, customer, or vendor organizations to participate as SecureZIP Partner partners. The sponsor supplies instructions on how to contact PKWARE to request a copy of the ***SecureZIP Partner*** application. After ***SecureZIP Partner*** is installed and configured at the partner location, sponsor and partner can exchange data files with confidence that the data is protected.

Co-existence with Other PKWARE Products

The ***SecureZIP Partner for IBM i*** product package can be installed alongside other ***Smartcrypt**i*** product releases but must be given a different library name.

Recommendations

- Installations using both ***SecureZIP Partner for IBM i*** and ***Smartcrypt for IBM i*** on the same system should configure separate local certificate stores for each. Although certificate store components can co-exist in the same Store, care must be taken that full-featured component names assigned by the system administrator do not conflict with names automatically generated by SecureZIP Partner.
- When other releases of ***Smartcrypt**i*** are operating in the same system, only one library should be in the job's library list.

SecureZIP Partner Certificate Store Administration and Configuration

Certificate administration and use in the ***SecureZIP Partner*** operating environment differ slightly from the case with full-featured ***Smartcrypt for IBM i***.

Whereas all digital key components are individually administered in a full-function installation, SecureZIP Partner components are pre-packaged for distribution and installation into a Sponsor Distribution Package. Many features of SecureZIP Partner work the same as in full-featured Smartcrypt, but some features work differently and

use special components of a Sponsor Distribution Package instead of standard Smartcrypt components.

The following table indicates which components of the ***Smartcrypt for z/OS*** local certificate store are used in relationship with the mode of operation.

Certificate Use	*Full Feature Smartcrypt*	*SecureZIP Partner*
Archive Signature Authentication	Full Certificate Store*	Sponsor Distribution Package AUTHCHK((*ARCHIVE *SPONSOR 'a0000100.p7'))
File Signature Authentication	"	Full Certificate Store
Archive Signing	"	Full Certificate Store
File Signing	"	Full Certificate Store
Encryption	"	Sponsor Distribution Package ENTPREC((*SPONSOR 'r0000100.p7'))
Decryption	"	Full Certificate Store

* A full-function, standard certificate store includes public-key and/or private-key X.509 certificate files along with their associated certificate authority trust chain and an optional certificate revocation list. To set up a certificate store, use the ***Smartcrypt for IBM i*** certificate store administration tool. You are responsible for obtaining the appropriate digital certificate resources.

Choosing a Configuration Model

Depending on your installation's business requirements for segregated process controls, you may choose to coordinate the operation of sponsor profiles from a centralized certificate store, or segregate the configurations entirely.

Components supporting a Sponsor profile are installed as stream files in a subfolder of the ../Sponsor folder.

Shared Certificate Store for Multiple Sponsor Profiles

The ***Smartcrypt for IBM i*** certificate store supports the ability to install and configure multiple sponsor profiles within a single store. This centralized approach may be the simplest to manage.

Configured Sponsor Package Components

When a Sponsor Distribution Package is installed, various components are configured within the SecureZIP Partner Sponsor folder. The following table describes the components and how they are used.

Component Folder Location	*Description*	*Usage*
/Sponsor/Package	The Package folder is where the Sponsor's issuing package is saved. When a Sponsor's package is received, it should be copied or Ftp'd to the folder so it can be installed.	This is the Sponsor Distribution Package, for example, pkware.dat for the test package. The PKPLINKIN command is used to install the package into your environment.

Component Folder Location	*Description*	*Usage*
/Sponsor/Info	The Info folder contains an external XML file of the Sponsor file's contents. This is used for listing the installed sponsors.	Files in this folder will be a XML files with a prefix of I followed by seven numeric digits of the sponsor's ID number. For example, the test sponsor file name is I0000000.xml. This is used for displaying the Sponsor's name, ID number and creation date, comments, etc. Use PKPLINKLST command to list the contents.
/Sponsor/Auth **(Read Mode Authorized Recipient File)**	The Auth folder contains a PKCS#7 file for each sponsor that contains a collection of end-entity certificates that *SecureZIP Partner for IBM i* should recognize as being valid signers for the sponsor. This store will also contain any CA certificates necessary to verify the trust chains of the end-entity certificates. *SecureZIP Partner for IBM i* will use this file to verify that the central directory of an archive to be extracted is properly signed and valid.	Files in this folder will be special public certificate files with a prefix of A followed by seven numeric digits of the sponsor's ID number. For example, for the test sponsor it will be A0000000.p7. These PKCS#7 files identify a list of authentication public-key/certificates used to validate the source of an input ZIP archive referred to by the Sponsor Authentication Configuration Setting supplied to the SecureZIP Partner run. Used with PKZIP and PKUNZIP command parameter AUTHCHK with a type *SPONSOR for the *ARCHIVE type. For example: AUTHCHK((*ARCHIVE *SPONSOR 'A0000000.p7')) or AUTHCHK((*ARCHIVE *SPONSOR 0))
/Sponsor/Recip **(Write Mode Authorized Recipient Files)**	The Recip folder contains a PKCS#7 file for each Sponsor containing a collection of end-entity certificates to be used as recipients when encrypting data for transmission back to the sponsor. This file will also contain any CA certificates necessary to verify the trust chains of the end-entity certificates. It will be internally signed by the special *SecureZIP Partner for IBM i* certificate.	Files in this folder will be a special public certificate files with a prefix of R followed by seven numeric digits of the sponsor's ID number. For example for the test sponsor it will be R0000000.p7. These PKCS#7 files identify a list of Sponsor-provided public-key/certificates that can be used to encrypt new data being added to a ZIP archive. Used with PKZIP command parameter ENTPREC with a type *SPONSOR. For example: ENTPREC (*SPONSOR 'R0000000.p7')) or ENTPREC (*SPONSOR 0)) Only one RECIPIENT configuration command will be accepted for processing per ZIP job.

Component Folder Location	*Description*	*Usage*
/Sponsor/Log	The Log folder contains the SecureZIP Partner_Install.log file used to time-stamp installation of each sponsor file.	During package installation with command PKPLINKIN, the log file SecureZIP Partner_Install.log will be updated with a time stamped entry of what was updated and by which User ID. Use the command WRKF or DSPF to browse this file.
/Sponsor/Temp	The Temp folder is used to install the Partner Sponsor's package. It is cleared at the start and end of each installation, so no permanent files should be saved in this folder.	Command PKPLINKIN uses the area to install the sponsor packages to the environment.

Installing a Sponsor Distribution Package

You must install a Sponsor Distribution Package to configure SecureZIP Partner to work with ZIP archives for, or from, a particular sponsor. Each sponsor has a different Sponsor Distribution Package.

Sponsor Distribution Package Installation Steps

A Sponsor Distribution Package is installed as a configuration to an existing local /Sponsor folder. The following steps define the process to configure SecureZIP Partner for operations with a related sponsor.

Note: It is highly recommended that you keep a copy of a Sponsor Distribution Package after you install it in case you later want to install to a certificate store having a different name or location.

To install a Sponsor Distribution Package, follow these steps:

1. Verify that the SecureZIP Partner Sponsor folders are created and defined with the command PKCFGSEC, parameter PLPATH. Refer to the section "PKPLINKIN - SecureZIP Partner Package Install Command" later in this chapter.
2. Perform a binary transfer of the Sponsor Distribution Package to the system. Place the package in the store folder ../Sponsor/Package.
3. Install the package with command ➔PKPLINKIN PLPACKAGE('package name') PLUPDATES(*NO). If the package will replace a package that is already installed, use PLUPDATES(*YES). The sponsor ID number is used to reference all commands and runs for this sponsor. Note this ID number for future use.
4. Use command ➔PKPLINKLST PLSPONID(*ALL) to view all sponsors and the sponsor files that have been installed. With *ALL, you can view all sponsors and associated sponsor ID numbers.

Sample PKWARE Sponsor Package

A sample Sponsor Package has been included and will be restored to the ../Sponsor/Package/PKWARE.dat as part of the startup procedure. This will assist you in understanding the process for Sponsor Package installation and to verify the certificate store setup.

1. Verify the test package has been restored with a ➔WRKLNK

'../Sponsor/Package/PKWARE.dat'.

Install the test package with ➔PKPLINKIN PLPACKAGE('PKWARE.dat') PLUPDATES(*NO). This will install sponsor ID number of 0.

```
PKPLINKIN - SecureZIP Partner Sponsor Package Installer
Installing Sponsor Package 'PKWARE.dat'
Sponsor ID Number = 0   'PKWARE, Inc.'
Created: 2005-09-28 12:57:44
Validated '/yourpath/PKWARE/PLstore/Sponsor/Temp/auth.p7' AUTH certificate.
Validated '/yourpath/PKWARE/PLstore/Sponsor/Temp/recip.p7' RECIP certificate.
Added Info File: 'I0000000.xml'
Added Auth File: 'A0000000.p7'
Added Recip File: 'R0000000.p7'
PKPLINKIN "/yourpath/PKWARE/PLstore/Sponsor/PACKAGE/PKWARE.DAT" Completed
Successfully
```

2. If you need to review the history of an update, see the log file.

   For example, you can view the updates by doing a DSF of the log file:

   ➔ DSPF STMF('/yourpath/PKWARE/PLstore/Sponsor/log/SecureZIP Partner_Install.log')

```
***** JOB(024237/EVWSS/EVWSSL01) *****
Sponsor Package - /yourpath/PKWARE/PLstore/Sponsor/PACKAGE/PKWARE.dat
Sponsor Name - PKWARE, Inc.
Install Date - 2005/10/05 11:24:03
Installed - /yourpath/PKWARE/PLstore/Sponsor/INFO/I0000000.xml
Installed - /yourpath/PKWARE/PLstore/Sponsor/AUTH/A0000000.p7
Installed - /yourpath/PKWARE/PLstore/Sponsor/RECIP/R0000000.p7
```

3. Use the SecureZIP Partner Package list administration command to view the installed package. Use ➔PKPLINKLST PLSPONID(*ALL) or ➔PKPLINKLST PLSPONID(0) to view the installed Sponsor configuration. The following entries should be displayed:

```
PKPLINKLST SecureZIP Partner Sponsor Listing------2005/09/16 07:56:43
Innnnnnn  Sponsor Creation date:xxxxxxxxxxxxx    Type=x
 Customer Name
 <Comments>
Number of Files nn: Annnnnnn, Rnnnnnnn, Cnnnnnnn

OR distributed package:
PKPLINKLST PatnerLink Sponsor Listing------2005/10/12 15:09:44
I0000000.xml  Sponsor Creation Date: 2005-09-28 12:57:44   Type = 1
 PKWARE, Inc.
 This sponsor distribution package is for testing purposes only!
Number of Files 2: A0000000.p7, R0000000.p7,

Press ENTER to end terminal session.
```

4. Modify and run the test job in Smartcrypt library with the name PLIVPZIP to verify the use of the test Sponsor configuration. The source for PLIVPZIP is included in the QCLSRC file.

Updating a Sponsor Distribution Package

A currently configured sponsor in the local Sponsor store can be updated with a newer version by following the normal steps for installing a Sponsor Distribution Package with the PKPLINKIN. If the parameter PLUPDATES is *NO, then no

replacements will take place. If the Sponsor file install run should replace a currently installed sponsor, the code PLUPDATES is *YES.

- The installation procedure will check the creation date (as contained in the XML data) of the input package against the previously installed package information.
- If the creation date of the input package is later than the previously installed package, then the old components will be removed, and the new package components installed.
- If the creation date of the input package is less than the currently installed package, the update replacement will fail. In order to install a sponsor package that is older than the current installed package, all associated files should be deleted. See the section “Removing a Sponsor Distribution Package” later in this chapter. After removing the package, the PKPLINKIN command can be rerun to install an older package.

Removing a Sponsor Distribution Package

To remove a currently installed package requires the manual removal of all associated files for that installed package.

1. After identifying the Sponsor ID number to be removed, issue the command PKPLINKLST PLSPONID(id) to review all files to be removed.
2. Do a RMVLNK ‘../Sponsor/Type/file name’ for all files.

   For Example:

   ➔RMVLNK ‘/mypath/Sponsor/Auth/A0000000.p7’
   ➔RMVLNK ‘/mypath/Sponsor/Recip/R0000000.p7’
   ➔RMVLNK ‘/mypath/Sponsor/Info/I0000000.xml’

Providing a Sponsor Configuration for Execution

The folders where the Sponsor Distribution Package components were installed must be provided (for Read access) to the executing Read mode (UNZIP) or Write mode (ZIP) jobs. In addition, specific configuration components will be required for the associated processing request.

SecureZIP Partner IVP Test Archive and CL Program

Included in the distribution library is a test archive that has been signed by the distributed test PKWARE sponsor ID number 0. This file is named PLIVPZIP and can be found in the Smartcrypt library. Also there is a program PLIVPZIP (source is in QCLSRC) that will run several tests to demonstrate simple test cases of successful and failing ZIP and Unzip jobs.

The following is a list of the steps in the CL program PLIVPZIP followed by sample displays of each step.

PLIVPZIP Step Review:

```
/* Program:   PLIVPZIP              Sample VERSION 16.0        */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*    Parameter  Program Variables                               */
/*      Input                                                   */
/* Smartcrypt Library  as input                                  */
/* EXAMPLE:                                                     */
/* CALL PGM(PLIVPZIP) PARM('PKW14061l')                         */
/*                                                              */
/*                                                              */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/**************************************************************/
/* Abstract: This is a sample IVP CL program that can be used to */
/*    verify initial install.  It uses the distributed archive   */
/*    that was signed with the PKWARE test sponsor package or    */
/*    Sponsor ID nbr 0.                                         */
/*    Assumption: PKWARE.dat sponsor package installed          */
/*    The CL program has one input variable the PKZIP library.   */
/*                                                              */
/**************************************************************/
/**************************************************************/
/* Processing tests                                             */
/*                                                              */
/* STEP NAME                                                    */
/* =========                                                    */
/* INITIALIZ Initialize variables clean up test files           */
/*                                                              */
/* VIEWFAIL Generate a detail View report of PKIVPZIP archive    */
/*     Purposely fail access to PLIVPZIP with no Sponsor File    */
/* - Expect AQZ0038 SecureUNZIP Completed with Errors           */
/* - AQZ0813 Archive:  PKW14061L/PLIVPZIP(PLIVPZIP), 4608 bytes, */
/*               1 file,, 1 Segment                             */
/* - AQZ0800 Filename: SECZIP/READER/README.TXT                 */
/* - AQZ0886 Archive has been Digitally Signed.                 */
/* - AQZ0353 No Partner files supplied for Reader authentication */
/*                                                              */
/* VIEWOK1 Generate a detail View report of PKIVPZIP archive     */
/*      Include valid Sponsor file for Sponsor ID 0 in AUTHCHK   */
/* - Expect AQZ0037 SecureUNZIP   Completed Successfully        */
/* - AQZ0813 Archive:  PKW14061L/PLIVPZIP(PLIVPZIP), 4608 bytes, */
/*               1 file,, 1 Segment                             */
/* - AQZ0800 Filename: SECZIP/READER/README.TXT                 */
/* - AQZ0886 Archive has been Digitally Signed.                 */
/* - AQZ0422 Archive was signed by "PKWARE PartnerLink TEST     */
/*           Signing Certificate" and verified                  */
/*                                                              */
/* AUTHTEST Use the configured Sponsor 0 to AUTHCHK PKIVPZIP for */
/*          (TEST) using passphrase 'PKWARE, Inc.'              */
/* - Expect AQZ0037 SecureUNZIP   Completed Successfully        */
/* - AQZ0422 Archive Directory Authentication Succeeded         */
/* - AQZ0250 tested okay SECZIP/READER/README.TXT               */
/*                                                              */
/* EXTRFAIL Attempt to extract a file from PKIVPZIP with no     */
/*          Sponsor AUTHCHK specified                           */
/* - Expect AQZ0037 SecureUNZIP   Completed Successfully        */
/* - AQZ0353 No Partner files supplied for Reader authentication.*/
/* - AQZ0036 SecureUNZIP  extracted     0 files                 */
/*                                                              */
/* EXTRACT Use the configured Sponsor 0 to AUTHCHK PKIVPZIP for  */
/*          (EXTRACT) using passphrase 'PKWARE, Inc.'           */
/*          extract to file TMPTEST in zip library              */
/*          Use DSPF FILE(ZIPLIB/TMPTEST) MBR(READMETXT)        */
/*          When finished file TMPTEST can be deleted           */
/* - Expect AQZ0037 SecureUNZIP   Completed Successfully        */
/*                                                              */
/* SLNKZIP Use the configured PKIVPPKG to create an archive     */
/*         using sponsor 0 for AES256 encryption, no signer     */
/* - Expect AQZ0020 Smartcrypt Completed Successfully           */
/* - AQZ0169 Digital Certificate Request List: Encryption       */
```

```
/*                                              Recipients         */
/* - AQZ0170  Rqrd    Pub  *SPONSOR                    -          */
/*            /yourpath/PKWARE/PLstore/Sponsor/RECIP/r0000000.p7 */
/* - AQZ0183 Encryption Recipients List-----------1 processed:   */
/* - AQZ0184 CN=PKWARE PartnerLink TEST Encryption Certificate   */
/*                                                               */
/* SLNKVIEW  Generate a detail View report of the new archive    */
/* - Expect AQZ0037 SecureUNZIP   Completed Successfully         */
/* - AQZ0888   Number Certificate Recipients 1                   */
/*                                                               */
/*****************************************************************/
```

Sample Display for step VIEWFAIL:

```
Archive:  PKW14061L/PLIVPZIP(PLIVPZIP), 4608 bytes, 1 file, 1 Segment
Archive Comment:"Smartcrypt for zSeries by PKWARE"
Filename: SECZIP/READER/README.TXT
Detected File type:     Text   Encrypt=Strong Encrypted
Created by:             PKZIP for zSeries(R)  10.0
Zip Spec to Extract:    5.1  Or Greater
Compression method:     Deflated   [Superfast]
Date and Time           2005 Sep 27 14:12:22
Compressed size:        198 bytes
Uncompressed size:      344 bytes
32-bit CRC value (hex): 5256afd4
Extended attributes:    yes, [Length = 3506]
Strong Encryption AES 256 Key.
Algorithm Key 256,  Security type Passphrase
Number Certificate Recipients 0
A subfield with ID 0x0014  (Certificate Store)  and 3266 data bytes
A subfield with ID 0x0016  (Archive Signature)  and 216 data bytes
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 1 file, 344 bytes uncompressed, 198 bytes compressed:   42%
Archive has been Digitally Signed.
No Partner files supplied for Read Mode authentication.
SecureUNZIP  extracted     0 files
SecureUNZIP Completed with Errors
```

Sample Display for step VIEWOK1:

```
Digital Certificate Request List:Archive Authenticator
Rqrd    Pub  *SPONSOR                    - a0000000.p7
Archive Authenticator List-----------1 processed:
Archive:  PKW14061L/PLIVPZIP(PLIVPZIP), 4608 bytes, 1 file, 1 Segment
Archive Comment:"Smartcrypt for zSeries by PKWARE"
Filename: SECZIP/READER/README.TXT
Detected File type:     Text   Encrypt=Strong Encrypted
Created by:             PKZIP for z/OS  10.0
Zip Spec to Extract:    5.1  Or Greater
Compression method:     Deflated   [Superfast]
Date and Time           2005 Sep 27 14:12:22
Compressed size:        198 bytes
Uncompressed size:      344 bytes
32-bit CRC value (hex): 5256afd4
Extended attributes:    yes, [Length = 3506]
Strong Encryption AES 256 Key.
Algorithm Key 256,  Security type Passphrase
Number Certificate Recipients 0
A subfield with ID 0x0014  (Certificate Store)  and 3266 data bytes
A subfield with ID 0x0016  (Archive Signature)  and 216 data bytes
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 1 file, 344 bytes uncompressed, 198 bytes compressed:   42%
```

```
Archive has been Digitally Signed.
Archive was signed by "PKWARE PartnerLink TEST Signing Certificate" and verified
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

Sample Display for step AUTHTEST:

```
Digital Certificate Request List:Archive Authenticator
Rqrd    Pub  *SPONSOR                    - a0000000.p7
Archive Authenticator List-----------1 processed:
UNZIP Archive: PKW14061L/PLIVPZIP(PLIVPZIP)
Archive Comment:"Smartcrypt for zSeries by PKWARE"
Searching Archive PKW14061L/PLIVPZIP(PLIVPZIP)  for files to extract
Archive was signed by "PKWARE PartnerLink TEST Signing Certificate" and verified
Testing: SECZIP/READER/README.TXT
SECZIP/READER/README.TXT tested OK
No errors detected in compressed data of PKW14061L/PLIVPZIP(PLIVPZIP).
SecureUNZIP   Completed Successfully
```

Sample Display for step EXTRFAIL:

```
UNZIP Archive: PKW14061L/PLIVPZIP(PLIVPZIP)
Archive Comment:"Smartcrypt for zSeries by PKWARE"
Searching Archive PKW14061L/PLIVPZIP(PLIVPZIP)  for files to extract
No Partner files supplied for Read Mode authentication.
SecureUNZIP  extracted     0 files
SecureUNZIP Completed with Errors
```

Sample Display for step EXTRACT:

```
Digital Certificate Request List:Archive Authenticator
Rqrd    Pub  *SPONSOR                    - a0000000.p7
Archive Authenticator List-----------1 processed:
UNZIP Archive: PKW14061L/PLIVPZIP(PLIVPZIP)
Archive Comment:"Smartcrypt for zSeries by PKWARE"
Searching Archive PKW14061L/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:PKW14061L/TMPTEST(READMETXT) Text
SecureUNZIP  extracted     1 files
SecureUNZIP   Completed Successfully
```

Sample Display for 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 PartnerLink TEST Encryption Certificate
EMail=PKWAREPartnerLinkCA@pkware.com
Found  1 matching files
Compressing PKW14061L/TMPTEST(READMETXT) in TEXT mode
Add  PKW14061.L/TMPTEST/READMETX.T  --  Deflating (69%)  encrypt(AES 256Key)
Smartcrypt Compressed 1 files in Archive PKW14061L/PLIVPZIP(NEWTESTZ)
Smartcrypt Completed Successfully
```

Sample Display for step SLNKVIEW:

```
Archive:  PKW14061L/PLIVPZIP(NEWTESTZ), 1536 bytes, 1 file, 1 Segment
Archive Comment:"Smartcrypt for IBM i"
Filename: PKW14061.L/TMPTEST/READMETX.T
Detected File type:     Text   Encrypt=Strong Encrypted
Created by:             PKZIP(R) for IBM i  10.0
Zip Spec to Extract:    6.1  Or Greater
Compression method:     Deflated   [Superfast]
Date and Time           2005 Oct 4 12:10:20
Compressed size:        708 bytes
Uncompressed size:      343 bytes
32-bit CRC value (hex): 3127928f
Extended attributes:    yes, [Length = 184]
Strong Encryption AES 256 Key.
Algorithm Key 256,  Security type Certificate Key
Number Certificate Recipients 1
Recipient List:
Record Length: 132
Lib Text Desc: Smartcrypt(R) for IBM i Lib-V16.0
File Text Desc: File created by PKZIP(R) for IBM i
Mbr Text Desc: Member created by PKZIP(R) for IBM i
Source or Data: PF-DTA
File Comment:"none"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found 1 file, 343 bytes uncompressed, 708 bytes compressed:   0%
SecureUNZIP  extracted     0 files
SecureUNZIP   Completed Successfully
```

SecureZIP Partner Commands

Two commands are provided to install and display SecureZIP Partner Sponsor Distribution Packages.

PKPLINKIN - SecureZIP Partner Package Install Command

The PKPLINKIN command installs a Sponsor Distribution Package into your PKWARE SecureZIP Partner system. In order to execute this command, the Sponsor Distribution Package must already be in place in the ../Sponsor/PACKAGE directory on your system. If the Sponsor Distribution Package is not in place, copy or FTP the file (in binary mode) into the ../Sponsor/PACKAGE directory before executing the PKPLINKIN command.

When executing the PKPLINKIN command, the Sponsor Distribution Package must be specified by name. In addition, the parameter must be set to allow or disallow Sponsor Distribution Package updates. Below is a screen shot of the prompted PKPLINKIN command. Press the PF4 key with the PKPLINKIN on the command line to display this screen. A detailed description of the PKPLINKIN command parameters follows.

```
                    SecureZIP Partner Package Installer (PKPLINKIN)

Type choices, press Enter.

Sponsor Package to Install . . .   PKWARE.dat
Allow Package Updates? . . . . .   *NO          *NO, *YES

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

PKPLINKIN Command Keyword Details

PLPACKAGE - Sponsor Package to Install

The name of the Sponsor Distribution Package to be installed. As stated previously, this file must already be in place in the ../Sponsor/PACKAGE directory in order for the command to succeed.

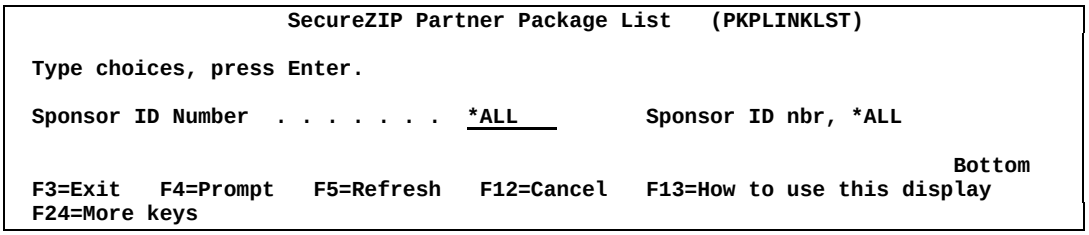
PLUPDATES - Allow Package Updates?

In normal operation of the PKPLINKIN command, installing a Sponsor Distribution Package with the same Sponsor ID number in not allowed. However, there may be valid cases such as an updated Sponsor Distribution Package or recovering from a system failure where reinstallation is required. In these cases, the PLUPDATES parameter can be set to *YES. With the *YES parameter set, the Sponsor Distribution Package will be installed over the existing version of the package.

***NO**	The default setting for the PLUPDATES parameter. In this case, an attempt to install a Sponsor Distribution Package that already exists will result in a failure.
***YES**	If the PLUPDATES parameter is set to *YES, the PKPLINKIN command will allow the installation of a Sponsor Distribution Package with a Sponsor ID matching an already installed Sponsor. If an existing Sponsor is found with a matching Sponsor ID, the user will receive a warning message, but the PKPLINKIN command will end successfully.

PKPLINKLST - List Installed SecureZIP Partner Sponsors Command

The PKPLINKLST command lists the information regarding Sponsor Distribution Packages already installed on the system. Below is a screen shot of the prompted PKPLINKLST command. Press the PF4 key with the PKPLINKLST on the command line to display this screen. A detailed description of the PKPLINKLST command parameters follows.



PKPLINKLST Command Keyword Details

PLSPONID- Sponsor ID Number

The PLSPONID parameter allows the user to pick the Sponsor Distribution Package that he or she wishes to receive information about. Enter the sponsor ID number of the sponsor of interest to display that sponsor. If the user wishes to see all the users or is unsure of the Sponsor ID, the *ALL option can be given for PLSPONID. In this case, information for all the Sponsor Distribution Packages installed on the system is displayed.

12 PKCRYRUN “Encryption/Hashing Facility Testing” Command

Requires Smartcrypt with IBM i OS V5R3M0 or higher

The ***Smartcrypt for iSeries*** IBM Cryptographic Facilities Integration feature enables the selection of locally activated IBM cryptographic facilities to complete cryptographic service requests for data encryption and digital signature processing.

Cryptographic Facility Categories

Smartcrypt for iSeries automatically determines which facilities are available for use when a cryptographic service is required. It also selects which facility to use based on configurable preference lists specified through either the defaults configuration (PKCFGSEC) or a parameter.

Facilities are organized into sets of similar cryptographic functionality. For example, all symmetric data encryption methods, such as DES and AES, fall into the ADVCRYPT facility category. Digital signature creation or authentication requires a cryptographic HASH facility. There are two facilities available: PKWARE’s Software APIs (PKSW), and IBM Software APIs (IBMSW).

PKCRYRUN Command Summary with Parameter Keyword Format

The PKCRYRUN utility command provided with the product can help the administrator or user select the most appropriate facility settings when planning to employ cryptographic features of ***Smartcrypt for iSeries***.

This order of precedence generally provides the best performance when used in conjunction with the default ADVCRYPT and SIGNER algorithm settings and ensures that at least one facility can be selected to complete the processing request.

Once the sequence of algorithms are determined for an environment, then these settings can be set for all runs using the PKCFGSEC command FACENC and FACHASH parameters.

PKCRYRUN can also be used to access the CPU usage of each encryption and hashing algorithms for PKWARE and the IBM’s API’s counterpart. When testing the algorithms it is best

to have them submitted to batch using the parameter TESTCD(*BATCH) since these test CPU intensive.

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 command process reference the IBM Home page for your version of the operating system.

```
RUNTYPE (   Processing Type             )
                  {*CRYPT }
                  {*HASH }

RPTMTX (   Algorithm Matrix Summary?             )
                  {*YES }
                  {*NO }

RPTMEGSM (   Meg/CP Sec. Summary?       )
                  {*YES }
                  {*NO }

RPTCPUSE (   CPU Usage Summary?         )
                  {*NO }
                  {*YES }

RPTELAP (   Elapse Time Summary?        )
                  {*NO }
                  {*YES }

RPTDTST (   Detail Test Report?         )
                  {*NO }
                  {*YES }

RPTSMTX (   Algorithm Availability Matrix?              )
                  {*NO }
                  {*YES }

TESTCD (   Run Algorithm Test?                  )
                  {*NO }
                  {*YES }
                  {*BATCH }

TESTBYTES (     Compare Test Results?           )
                  {0-1000 }


COMPCD (   Test Compare?                )
                  {*NO }
                  {*YES }

FACENC (   Facility:Encryption          )
                  { PKSW_IBMSW }
                  { IBMSW_PKSW }
                  { PKSW }
                  { IBMSW }

FACHASH (   Facility:Hashing            )
                  { PKSW_IBMSW }
                  { IBMSW_PKSW }
                  { PKSW }
                  { IBMSW }

FIPSCRL (   FIPS Controls         )
                  {*NO }
```

```
                    {*USEREQ}
                    {*USEAVAIL}
                    {*FORCE}


RPTDMP (   Special Dump Details?        )
                    {*NO }
                    {*YES }
```

PKCRYRUN Command Keyword Details

RUNTYPE - Processing Type

RUNTYPE (*CRYPT|*HASH)

Specifies which type of algorithm to run.

*CRYPT will run the available encryption algorithms.

*HASH will run the available hashing algorithms.

RPTMTX - Algorithm Matrix Summary?

RPTMTX (*YES|*NO)

Show the “Algorithm Matrix Summary” (See Sample Report 6).

*YES will show/display report.

*NO will not show/display the report.

RPTMEGSM - Meg/CP Sec. Summary?

RPTMEGSM (*YES|*NO)

If TESTCD is *YES or *BATCH to test the algorithms then this parameter will show the “Meg/CP Sec. Summary” (See Sample Report 5).

*YES will show/display report.

*NO will not show/display the report.

RPTCPUSE - CPU Usage Summary?

RPTCPUSE (*NO|*YES)

If TESTCD is *YES or *BATCH to test the algorithms then this parameter will show the “CPU Usage Summary” (See Sample Report 3).

*NO will not show/display the report.

*YES will show/display report.

RPTELAP - Elapse Time Summary?

RPTELAP (*NO|*YES)

If TESTCD is *YES or *BATCH to test the algorithms, then this parameter will show the “Elapse Time Summary” (See Sample Report 4).

*NO will not show/display the report.

*YES will show/display report.

RPTDTST Detail Test Report?

RPTDTST (*NO|*YES)

If TESTCD is *YES or *BATCH to test the algorithms, then this parameter will show the “Detail Test Report” (See Sample Report 2).

*NO will not show/display the report.

*YES will show/display report.

RPTSMTX - Algorithm Availability Matrix?

RPTSMTX (*NO|*YES)

This parameter will show the facility control matrix that is built, better known as the “Algorithm Availability Matrix” (See Sample Report 1).

*NO will not show/display the report.

*YES will show/display report.

TESTCD - Run Algorithm Test?

TESTCD (*NO|*YES |*BATCH)

Specify to whether to run testing phase of each algorithm.

*NO - will not run the algorithm testing.

*YES - will test each algorithm in interactive mode. This is NOT suggested since these runs are very CPU intensive.

*BATCH - will submit the running of the program to batch using the submit job command: SBMJOB CMD(***) JOB(PKCRYUTL).

TESTBYTES - Compare Test Results?

TESTBYTES (1| 0-1000)

Specifies the maximum numbers of megabytes to process with the algorithm in the testing phase. If COMPCD(*YES) is specified, then the maximum value of TESTBYTES is 10.

The actual maximum size for this parameter is 1000 megabytes.

COMPCD - Compare Test Results?

COMPCD (*NO|*YES)

For encryption a special test can be run to decrypt the encrypted data and compare the initial clear text to the decrypted clear text. When this option is set to yes then the maximum bytes to test is set to 10 megabytes.

*NO Do not decrypt and compare.

*YES Decrypt the encrypted data and compare to initial text.

FACENC - Facility:Encryption

FACENC (PKSW_IBMSW | IBMSW_PKSW | IBMSW | PKSW)

Sets the sequence or selection of which facilities to use for encryption runs.

PKSW_IBMSW – Facility API Sequence of PKWARE Software, IBM Software.

IBMSW_PKSW – Facility API Sequence of IBM Software , PKWARE Software.

IBMSW – Facility API Sequence of PKWARE Software.

PKSW – Facility API Sequence of IBM Software.

FACHASH - Facility:Hashing

FACHASH (PKSW_IBMSW | IBMSW_PKSW | IBMSW | PKSW)

Sets the sequence or selection of which facilities to use for hashing runs.

PKSW_IBMSW – Facility API Sequence of PKWARE Software, IBM Software.

IBMSW_PKSW – Facility API Sequence of IBM Software , PKWARE Software.

IBMSW – Facility API Sequence of PKWARE Software.

PKSW – Facility API Sequence of IBM Software.

FIPSCRL - FIPS Controls

FIPSCRL (*NO| *USEREQ | *USEAVAIL | *FORCE)

Not Available. Reserved for future use.

RPTDMP - Reports:Dump Details?

RPTDMP (*NO|*YES)

Not Available. Reserved for future use.

Sample PKCRYRUN Command Screen:

```
                Smartcrypt Crypto Utility 16.0 (PKCRYRUN)
Processing Type  . . . . . .   *CRYPT        *CRYPT, *HASH
Algorithm Matrix Summary?  . . .   *YES          *NO, *YES
Meg/CP Sec. Summary?   . . . . .   *YES          *NO, *YES
CPU Usage Summary?     . . . . .   *NO           *NO, *YES
Elapse Time Summary?   . . . . .   *NO           *NO, *YES
Detail Test Report?  . . . . . .   *NO           *NO, *YES
Algorithm Availability Matrix?     *NO           *NO, *YES
Run Algorithm Test?  . . . . . .   *NO           *NO, *YES, *BATCH
Megabytes to test  . . . . . .   1              Nbr of Megabytes for Testing
Compare Test Results?  . . . . .   *NO           *NO, *YES
Facility:Encryption  . . . . . .   PKSW_IBMSW    PKSW_IBMSW, IBMSW_PKSW...
Facility:Hashing . . . . . .   PKSW_IBMSW    PKSW_IBMSW, IBMSW_PKSW...
FIPS Controls  . . . . . .   *NO           *NO, *USEREQ, *USEAVAIL...
Special Dump Details?  . . . . .   *NO           *NO, *YES
```

Or command:

PKCRYRUN RUNTYPE(*CRYPT) RPTMEGSM(*YES) RPTMTX(*YES)
TESTCD(*BATCH) TESTBYTES(1)
FACENC(PKSW_IBMSW) FACHASH(PKSW_IBMSW)

PKCRYRUN Sample Reports

1. RPTSMTX "Algorithm Availability Matrix"

```
ZPEN340I /--------Encryptdata      Matrix (01)  --------/
ZPEN341I 0001(96 Bit Encryption )   Select (10/10) Smartcrypt
ZPEN341C   Status-IBMHW(-NoFacl-)  IBMSW(-NotCap-) PKW(  PKW   )
ZPEN341I 6801(RC4 Encryption    )   Select (10/10) Smartcrypt
ZPEN341C   Status-IBMHW(-NoFacl-)  IBMSW(-NotCap-) PKW(OPSSL )
ZPEN341I 660E(AES 128 Encryption)   Select (20/70) IBM Software
ZPEN341C   Status-IBMHW( -NoAPI-)  IBMSW(IBM API ) PKW(OPSSL)
ZPEN341I 660F(AES 192 Encryption)   Select (20/70) IBM Software
ZPEN341C   Status-IBMHW( -NoAPI-)  IBMSW(IBM API ) PKW(OPSSL)
ZPEN341I 6610(AES 256 Encryption)   Select (20/70) IBM Software
ZPEN341C   Status-IBMHW( -NoAPI-)  IBMSW(IBM API ) PKW(OPSSL)
ZPEN341I 6603(3DES Encryption   )   Select (20/70) IBM Software
ZPEN341C   Status-IBMHW( -NoAPI-)  IBMSW(IBM API ) PKW(OPSSL)
```

```
ZPEN341I 6601(DES Encryption    )   Select (20/70) IBM Software
ZPEN341C   Status-IBMHW( -NoAPI-)  IBMSW(IBM API ) PKW(OPSSL)
ZPEN374I-Completing with Return Code=0 Error Code=0-------------------------
```

Interpretation:

```
   ZPEN341I [alg_id]([algorithm_name]) Select ([code]) [Facility Category]

                  A request was made to report on the available
                  cryptographic facilities for the current operating
                  environment.

                  A separate report line is listed for each algorithm to
                  indicate which (if any) API is selected for use by
                  Smartcrypt after dynamic evaluation.

                  Each algorithm is validated against requested FACILITY
                  Settings and licensing.

                  [Facility Category]
                    The final facility chosen is shown.

                    NONE FOUND
                     No viable facility could be identified for use.
                     This algorithm cannot be serviced with the current
                     configuration.

                    IBM Hardware
                     The CryptoAPI identified in ZPEN321I (HW) will be used
                          Not active at this time.

                    IBM Software
                     The CryptoAPI identified in ZPEN321I (SW) will be used

                    PKWARE
                     The CryptoAPI identified in ZPEN321I (PKW) will be used
```

2. RPTDTST "Detail Test Report"

```
           ******************************************
ZPEN370I   *************Start of Testing*****************
           *************Nbr of Bytes=1048571*************
           *************Nbr of MEG=  1*******************

ZPEN361I Encryptdata      Matrix (01)
ZPEN362I (0001)96 Bit Encryption  Select (10)
ZPEN381I 96 Bit Encryption for Smartcrypt    Not Tested

ZPEN362I (6801)RC4 Encryption     Select (10)
ZPEN375I RC4 Encryption for Smartcrypt    Test CPU 0.012000 sec.

ZPEN362I (660E)AES 128 Encryption Select (20)
ZPEN375I AES 128 Encryption for IBM Software Test CPU 0.019000 sec.
ZPEN375I AES 128 Encryption for Smartcrypt    Test CPU 0.021000 sec.

ZPEN362I (660F)AES 192 Encryption Select (20)
ZPEN375I AES 192 Encryption for IBM Software Test CPU 0.022000 sec.
ZPEN375I AES 192 Encryption for Smartcrypt    Test CPU 0.025000 sec.

ZPEN362I (6610)AES 256 Encryption Select (20)
```

```
ZPEN375I AES 256 Encryption for IBM Software Test CPU 0.025000 sec.
ZPEN375I AES 256 Encryption for Smartcrypt    Test CPU 0.028000 sec.
ZPEN362I (6603)3DES Encryption    Select (20)
ZPEN375I 3DES Encryption for IBM Software Test CPU 0.267000 sec.
ZPEN375I 3DES Encryption for Smartcrypt    Test CPU 0.150000 sec.

ZPEN362I (6601)DES Encryption     Select (20)
ZPEN375I DES Encryption for IBM Software Test CPU 0.104000 sec.
ZPEN375I DES Encryption for Smartcrypt    Test CPU 0.051000 sec.

ZPEN385I-Testing Completed  Total CPU Seconds(0.731) Elapsed Seconds(4.141)
ZPEN374I-Completing with Return Code=0 Error Code=0-------------------------
```

Interpretation:

```
ZPEN362I show the starting of each algorithm group.

ZPEN375I will show the completions of each facility in each algorithm group along with the
amount CPU seconds consumed to process the test data.
```

3. RPTCPUSE "CPU Usage Summary"

```
    Test Summary Results          CPU Usage
 ZPEN383I   Crypto Facilities       HW          SW       OPSSLOPSSL/PKW
 ZPEN384I  96 Bit Encryption        N/A         N/A         N/A
 ZPEN384I  AES 128 Encryption      ------       0.019*      0.021
 ZPEN384I  AES 192 Encryption      ------       0.021*      0.025
 ZPEN384I  AES 256 Encryption      ------       0.024*      0.028
 ZPEN384I  3DES Encryption         ------       0.267*      0.151
 ZPEN384I  DES Encryption          ------       0.104*      0.051
 ZPEN384I  RC4 Encryption          ------      ------       0.012*
 ZPEN384I  AE-2    Decryption       N/A         N/A         N/A
```

Interpretation:

This report shows the number CPU seconds consumed during the process of each algorithm's test data. These times do not include the decryption or comparing.

```
  ZPEN384I   [crypto_algorithm]   [hw_API]  [sw_API]  [SecureZIP_API]

                  A request was made to produce a timing report for
                  supported cryptographic facilities in the current
                  operating environment.

                  A value will be listed for each facility category
                  associated with the correlated facility API listed in
                  ZPEN321I.

                  An "*" following a timing value indicates that the
                  corresponding API will be selected based on the
                  facility preference list shown in ZPEN322I.

                  A preceding header line will indicate whether this report
                  is for raw CPU time, or a computed throughput rate
                  in megabytes per CP Second.
```

```
                  Note:  The "96 bit encryption" algorithm will not have
                         timings run.  The SecureZIP(PKW) facility API
                         will always be selected for use when
                         ADVCRYPT(ZIPSTD) is specified.
```

4. RPTELAP "Elapsed Time Summary"

```
   P
    Test Summary Results          Elapsed Time Seconds
ZPEN383I   Crypto Facilities       HW          SW       OPSSLOPSSL/PKW
ZPEN384I  96 Bit Encryption        N/A         N/A         N/A
ZPEN384I  AES 128 Encryption      ------      0.100 *     0.110
ZPEN384I  AES 192 Encryption      ------      0.111 *     0.130
ZPEN384I  AES 256 Encryption      ------      0.121 *     0.148
ZPEN384I  3DES Encryption         ------      2.599 *     1.170
ZPEN384I  DES Encryption          ------      0.549 *     1.730
ZPEN384I  RC4 Encryption          ------      ------      0.068 *
ZPEN384I  AE-2    Decryption       N/A         N/A         N/A
```

Interpretation:

See ZPEN384I message in report 3. The time in this case represents the number elapsed seconds. On fast CPUs and with a small amount of data, this value will not be very significant.

5. RPTMEGSM "Meg/CP Sec. Summary"

```
ZPEN350I PKCRYUTL 2.0 Cryptographic API Review Utility
ZPEN350I Copyright (C) 1989-2016 PKWARE, Inc. All rights reserved.
ZPEN350I Program and Output used by permission only. PKWARE, Inc.
ZPEN351I PKCRYUTL Registered copy
ZPEN379I Decryption/Compare Results Active
ZPEN378I Testing with 1048571 Bytes Active

  Test Summary Results          Megabytes/CP Second
ZPEN383I   Crypto Facilities       HW          SW       OPSSLOPSSL/PKW
ZPEN384I  96 Bit Encryption        N/A         N/A         N/A
ZPEN384I  AES 128 Encryption      ------      52.64*      45.46
ZPEN384I  AES 192 Encryption      ------      47.62*      40.00
ZPEN384I  AES 256 Encryption      ------      41.67*      35.72
ZPEN384I  3DES Encryption         ------       3.75*       6.67
ZPEN384I  DES Encryption          ------       9.53*      19.61
ZPEN384I  RC4 Encryption          ------      ------      83.34*
```

Interpretation:

See ZPEN384I in report 3. The values in this case represent the calculated number of megabytes per CPU seconds.

6. RPTMTX "Algorithm Matrix Summary"

```
ZPEN320I CryptoAPI Facilities   HW   SW   Smartcrypt
ZPEN321I   96 Bit  Encryption   ---  ---    PKW
ZPEN321I   AES 128 Encryption   ---  Q3c  OPSSL
ZPEN321I   AES 192 Encryption   ---  Q3c  OPSSL
```

```
ZPEN321I   AES 256 Encryption   ---  Q3c  OPSSL
ZPEN321I   3DES    Encryption   ---  Q3c  OPSSL
ZPEN321I   DES     Encryption   ---  Q3c  OPSSL
ZPEN321I   RC4     Encryption   ---  ---  OPSSL
ZPEN321I   AE-2    Decryption   ---  ---  PKW +
ZPEN321I   CAST5#  Encryption   ---  ---  OPSSL
ZPEN321I   IDEA#   Decryption   ---  ---  OPSSL
ZPEN321I   CRC32   Hashing      ---  ---    PKW
ZPEN321I   SHA1    Hashing      ---  OWH  OPSSL
ZPEN321I   MD5     Hashing      ---  OWH  OPSSL
ZPEN321I   SHA256  Hashing      ---  OWH  OPSSL
ZPEN321I   SHA384  Hashing      ---  OWH  OPSSL
ZPEN321I   SHA512  Hashing      ---  OWH  OPSSL
ZPEN321I   Random Data Gen      ---  ---    PKW
ZPEN321I   Signing              ---  ---  OPSSL
ZPEN322I Facility Encryptdata      Seq: IBMHW(0) IBMSW(1) PKW(2)
ZPEN322I Facility Hash (Signature) Seq: IBMHW(0) IBMSW(1) PKW(2) OPSSLOPSSLOPSSLOPSSLOPSSL
```

Interpretation:

```
ZPEN321I   [crypto_algorithm]   [hw_API]  [sw_API]  [SecureZIP_API]

                    A request was made to report on the available
                    cryptographic facilities for the current operating
                    environment.  A separate report line is listed for
                    each algorithm to indicate which (if any) API is
                    available for use by Smartcrypt before dynamic evaluation.
                    A subsequent check of each algorithm will be performed
                    based on run-time options and environmental
                    characteristics.

                    [crypto_algorithm]
                      The [crypto_algorithm] name will also identify the use
                      type for the algorithm.

                     Symmetric Data Encryption algorithms:
                      96 Bit  Encryption
                      AES 128 Encryption
                      AES 192 Encryption
                      AES 256 Encryption
                      3DES    Encryption
                      DES     Encryption
                      RC4     Encryption
                      AE-2    Decryption
                      CAST5#  Encryption
                      IDEA#   Decryption

                           # indicates the algoritm is only avaiable for OpenPGP.

                     Data Integrity and Digital Signature algorithms:
                      CRC32  Hashing
                      SHA1   Hashing
                      MD5    Hashing
                      SHA256 Hashing
                      SHA384 Hashing
                      SHA512 Hashing

                    [hw_API]
                    [sw_API]
                    The IBM Cryptographic facilities are accessed through
                    one of the following APIs:

                      Q3c - Qc3EncryptData/Qc3DecryptData Encipher/Decipher
                      OWH - Qc3CalculateHash One way hash
```

```
[SecureZIP_API]
Smartcrypt provides software algorithms using one of the
following services:

 OPSSL - OpenSSL APIs
 PKW   - PKWARE internal routine
 PKW  +- PKWARE Specialized Decryption routines

"------" indicates that no service facility could be
identified under the API service category for the
algorithm.
```

13 Utility Programs

There are two utility programs to assist evaluating ZIP archives and OpenPGP files.

Using Zip Archive Scanning Utility

Smartcrypt offers a utility program PKSCNZIP to scan ZIP archives that can assist in evaluating if an archive is broken. This usually is run at the request of technical support to provide a summary of tags and headers in a ZIP archive.

Please note – The data will not displayed nor compromised. The output of the program will not be any part of the data nor encryption information.

The format and parameters of the PKSCNZIP is:

1. 'V' to show zip archive header data types only.
2. The full paths of the archive. If the file is stored in the QSYS library the use the IFS format for QSYS libraries.

Examples:

CALL PKSCNZIP ('V' '/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/MYMBR.MBR')

CALL PKSCNZIP ('V' '/myrootpath/mypath/myarchive.zip')

Sample Ouput:

```
PKWARE IBM i File Scan 01 - Copyright. 1989-2016 by PKWARE, Inc.
Valid Types only
In file: /pkzshare/bills/cos2Certstore.zip Using Standard fread etc
 size  :   10024
First 64 Bytes <504B03041400060008001C5D3838A81E956B4E110000F34C0000240090005
037425F312E45565753532E233230343535312E515052494E542E46303030312E50>
0001 buffer = 0000000000 <0000000000000000>  504b0304  Local Hdr
Local Dir<504B03041400060008001C5D3838A81E956B4E110000F34C000024009000>
0002 buffer = 0000004640 <0000000000001220>  504b0304  Local Hdr
Local Dir<504B03041400060008008062383801E80FD93F1200005951000024009000>
0003 buffer = 0000009521 <0000000000002531>  504b0102  Central Hdr
Central Dir<504B0102641214000600080001C5D3838A81E956B4E110000F34C0000240090000
00000000000000000000000000005037425F>
0004 buffer = 0000009747 <0000000000002613>  504b0102  Central Hdr
Central Dir<504B0102641214000600080008062383801E80FD93F120000595100002400900000
00000000000000000002012000005037425F>
0005 buffer = 0000009973 <00000000000026F5>  504b0506  End Central Hdr
End Central <504B0506000000000200200C4010000312500001D00>
EOF found: There is no error.
P_BYTE=6
buffers read = 0000010024 <0000000000002728>
```

Using OpenPGP Scanning Utilities

Smartcrypt offers two utilities to review OpenPGP structures. PKSCNPG program helps you examine the packet structure of an OpenPGP file. The PKQRYCDB command will display the contents of OpenPGP keyrings.

Scanning OpenPGP File Structures 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.

Packet Type	Description	Notes
1	Public-Key Encrypted Session Key	Wraps around a packet encrypted with a public/private key pair.
2	Signature	Digitally signed packet
3	Symmetric-Key Encrypted Session Key	Wraps around a passphrase-encrypted packet
4	One-Pass Signature	Precedes the signed data and contains enough information to allow the receiver to begin calculating any hashes needed to verify the signature. It allows the Signature packet to be placed at the end of the message, so that the signer can compute the entire signed message in one pass.
5	Secret Key	Contains both public and private key material
6	Public Key	Public key information; starts every OpenPGP file
7	Secret-Subkey	Contains both public and private key material for a subkey
8	Compressed Data	A literal data packet containing compressed data
9	Symmetrically Encrypted Data	Passphrase-encrypted data
11	Literal Data	Message body
12	Trust	Used only within keyrings, this packet specifies trustworthy key holders
13	User ID	Name and email address of the key holder
14	Public-Subkey	Contains public key material for a subkey
17	User Attribute	Additional information about the key holder not contained in UserID

Packet Type	Description	Notes
18	Symmetrically Encrypted and Integrity Protected Data	Passphrase-encrypted data that has been determined not to have changed since it was encrypted. This packet is used in combination with the Modification Detection Code packet (19).
19	Modification Detection Code	Contains a SHA-1 hash of plaintext data that matches the one included in the Symmetrically Encrypted and Integrity Protected Data packet (18).

The format and parameters of the PKSCNZIP is:

1. 'V' to show zip archive header data types only.
2. The full IFS paths of the OpenPGP file.
   Example:Call PKSCNPGP ('V' '/pkzshare/PGPtest/ PGP001PGPrecip.pgp')

Sample Report – Passphrase encrypted file

```
PKWARE z/OS PGP File Scan 01 - Copyright. 1989-2016 by PKWARE, Inc.
 Valid Types only
 In file: DD:MYDD01
  size  :   0
 First 64 Bytes <C31E0407030ADF312D2E20AFB248A82A4D5658B0EDD3A15BED6DEFC7F4
 8C3205610901>
 Packet  3--Symmetric-Key Encrypted Session Key # 1 L= 30 Final=1
 Packet 18--Sym. Encrypted & Integrity Protected Data # 2 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data # 3 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data # 4 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data # 5 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data # 6 L=4096 Final=0
 Packet 18--Sym. Encrypted & Integrity Protected Data # 7 L=4096 Final=0
 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=1476 Final=1

CPGPSCAN Total found Packet count=16 Len=58869
    1 Total- 3--Symmetric-Key Encrypted Session Key
   15 Total-18--Sym. Encrypted & Integrity Protected Data
CPGPSCAN - buffers read = 58869 <000000000000E5F5>
CPGPSCAN Ending.
```

Sample Report – Truncated Passphrase encrypted file

In this sample, the OpenPGP file was found to be incomplete (possibly due to an incomplete transfer) as indicated by a lack of a Final=1 Protected Data packet.

```
PKWARE z/OS PGP File Scan 01 - Copyright. 1989-2016 by PKWARE, Inc.
Valid Types only
In file: DD:MYDD01
 size  :   0
First 64 Bytes <C3260402030AE815312ECD58C030A847339C7DD53D2E30F2C7241690A9
Packet  3--Symmetric-Key Encrypted Session Key # 1 L= 38 Final=1
 Packet 18--Sym. Encrypted & Integrity Protected Data # 2 L=4096 Final=0
GetPGPfromBuff:  EOF(0/0)  found -owner<PGP> use=2  PGIinsw=1
skipPacket-18: Problem End of data during Skip hdr->pktdataleft

CPGPSCAN Warning  Warning last Packet did not have a final setting
```

```
CPGPSCAN Total found Packet count=2 Len=4138
   1 Total- 3--Symmetric-Key Encrypted Session Key
   1 Total-18--Sym. Encrypted & Integrity Protected Data
CPGPSCAN - buffers read = 1544 <0000000000000608>                    CPGPSCAN Ending
```

Scan an OpenPGP Keyring with PKQRYCDB command

As noted in the PKQRYCDB command chapter, this utility can also open and display the contents of public and private OpenPGP keyrings with parameter FTYPE(*PGPKRF). Of course the private key is not opened nor compromised at any time during this query operation.

Public Keyring Report (ref. “gpg –k”)

```
PKQRYCDB QUERY Smartcrypt Cert DataBase starting------2013/02/17 09:27:15
PKSCANCRT 005I scan(20) file is: /pkzshare/CStores/PGPKeys/pubring.pkr
PKSCANCRT 008I OpenPGP KeyRing #0 found (1536)
/pkzshare/CStores/PGPKeys/pubring.pkr 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

PKQRYCDB Run Totals for Library PKW14961S:
   Total Records In Error  =0
   Total Records Processed =2
PKQRYCDB Scan ending------
PKQRYCDB   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 iSeries 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 iSeries 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 iSeries 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 iSeries 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 iSeries 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 iSeries 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 8% more CPU than running with no encryption.. AES 128 on average could use around 8% more CPU than running with no encryption. AES 256 averages about 2% more usage when compared with AES 128 (or around 9.5% 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. ***Smartcrypt***^{i} 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 Memory Errors

In some rare cases, ***Smartcrypt***^{i} may experience a memory error condition. While running ***Smartcrypt***^{i} the program allocates memory for sorts, buffers, and other storage requirements. There are diagnostic checks in the system to validate memory requests. If a request fails, the job will terminate and send a message to the message queue indicating a memory allocation failure. Some systems limit the memory allocation for certain user/jobs which may cause this problem. In some cases, programs can have what are called “memory leaks” where memory is never freed. Signing off or ending the job will free the allocated memory. If a message indicates that a memory allocation error occurred, retrieve the failed job log and contact the Product Services Division at 1-937-847-2687 for assistance.

C

External Name Conversion Program CVTNAME

***Smartcrypt**i* provides an exit that calls a compiled CLP program named CVTNAME. PKZIP can change the names of iSeries file names to a ZIPPED file name to be used in the archive. PKUNZIP can change a ZIPPED file name in an archive to an iSeries file name. This is activated by using the parameter CVTFLAG and not setting the parameter to *NONE. ***Smartcrypt**i* will call the first CVTNAME program found in the library list.

***Smartcrypt**i* will call the program and pass three fields to CVTNAME and will expect a return field. The first field passed is a 256-byte field that contains the input name (in PKZIP, it is the iSeries name, and in PKUNZIP, it is the ZIPPED name in the archive) that can be changed. The second field passed is a 5-byte field that can be used to help code specific logic to control the name change process. The third field is up to 256 bytes of extended data from the command to provide more flexibility in controlling the conversion of file names.

The program will return up to 256 bytes of a file name that will be used as the output name in the archive.

The exit can be divided into different conversion routines by assigning each routine a five-character name, which is passed in as the CVTFLAG parameter value. This routine works for both the PKZIP and PKUNZIP programs, but normally you will need a different conversion routine or CVTFLAG for archiving or extraction. When the CVTNAME returns, the returned name should either hold the new name, or remain blank to indicate that the default conversion rules should be used.

Note: The PKZIP and PKUNZIP programs perform no validity checking on the name that is passed back and assumes that the name is valid and unique. If the name is not valid or unique, open errors or missing files may occur.

For PKZIP, the name exit receives the IBM i OS file name and the returned name is the name that will be used for the file name in the archive. Ensure that names returned are unique; otherwise, only the first of the duplicated files will be compressed.

For PKUNZIP, the CVTNAME exit receives the name of the file that is in the archive and expects to return the name of a file on the IBM i OS as ‘library/file(member)’. If this is invalid, then file open errors will occur. If they are valid but did not exist before, PKUNZIP will create the library, file, and member.

In the supplied library, there is a sample CVTNAME CLP that contains logic for the following flags:

- If the flag is *400, there is no conversion taking place, and the input name is passed back as a new name.
- If the flag is O4MS, the program will convert an iSeries file name “library/file(member)” to a MS-DOS name with ‘/’ (such as “library/file/member”).
- If the flag is MSO4, the program will take a MS-DOS file name “/dir1/dir2/dir3/name.xxx” and make it an iSeries name (such as “dir2/dir3{namexxx)”) where dir2 will be the library, dir3 will be a file name, and namexxx will be up to a 10-byte character member name.
- If the flag is *MSD, the program will convert an iSeries file name "library/file(member)" to a MS-DOS name with suffix .TXT to the file "library/file.TXT".
- If the flag is *MSM, the program will convert an iSeries file name "library/file(member)" to an MS-DOS name with a suffix .TXT added to the member name, such as "library/file/member.TXT".
- If the flag is *IFS, the program will format a file name "library/file(member)" for the iSeries QSYS.LIB integrated file system. For example "/QSYS.LIB/LIBRARY.LIB/FILE.FILE/MEMBER.MBR".
- If the flag is SPLF1, the program will take a spool file name "jobname/useriud/jobnumber/splfname/splfnbr.suffix" and build each in its own DCL field. Then taking the first 10 bytes of the CVTDATA passed from the command will build a new name that looks like “CVTDATA.splfnbr.suffix”.
- Of course the correct flag should be used with the appropriate file system.

Changes can be made to CVTNAME to fit the customer’s site requirements. The use of (and changes to) the CVTNAME program is the customer’s responsibility.

The following is an example of the CVTNAME CLP source code included in the ***Smartcrypt***^{i} library. (Note that the CVTNAME program could be in other languages such as RPG, C, and so on.)

Sample CVTNAME

```
/* Program:   CVTNAME              Sample VERSION 16.0          */
/*************************************************************/
/* Abstract: This is a sample CL program that can be used by     */
/*    PKZIP for IBM i product as an exit program to change the */
/*    names from iSeries to an archive name and visa versus.     */
/*    There are 4 parameters:                                    */
/*     1. The input name that is supplied by PKZIP or PKUNZIP    */
/*        to this program to analyze (up 255 bytes).             */
/*     2. The Name that this program can change and passes back  */
/*        to the calling programs to use (up to 255 bytes).      */
/*        If this is passed back with the 1st position blank,    */
/*        PKZIP/PKUNZIP will revert back to settings of the      */
/*        CVTTYPE parameter.                                     */
/*     3. A 5 byte field that represents a control field or      */
/*        or flags, that controls how CVTNAME should process the */
/*        name.                                                  */
/*     4. A 255 byte field that is user defined in the command to*/
/*        provide more information to assist controlling the     */
/*        logic in processing the name.  For Example it may      */
```

```
/*       time stamp prefix of the file names                    */
/*                                                              */
/*     This sample CVTNAME CLP has three flags accepted:        */
/*                                                              */
/*     1.  If the flag is *400 there are no conversion taking   */
/*     place and the input name is passed back as new name.     */
/*     2.  If the flag is O4MS,  the program will convert an    */
/*     iSeries name "library/file(member)"  to a MS-DOS         */
/*     name with '/' such as  "library/file/member".            */
/*     3.  If the flag is MSO4, the program  will take a MS-DOS */
/*     file name  "/dir1/dir2/dir3/name.xxx"  and make it an    */
/*     iSeries name such as  "dir2/dir3{namexxx)"  where dir2   */
/*     will be the library, dir3 will be a file name and        */
/*     namexxx will be up to a 10 byte character                */
/*     member name.                                             */
/*     4.  If the flag is *MSD,  the program will convert an    */
/*     iSeries file name "library/file(member)"  to a MS-DOS    */
/*     name with suffix .TXT to the file "library/file.TXT".    */
/*     5.  If the flag is *MSM,  the program will convert an    */
/*     iSeries file name "library/file(member)"  to a MS-DOS    */
/*     name with suffix .TXT to the member name                 */
/*     such as "library/file/member.TXT".                       */
/*     6.  If the flag is *MST,  the program will convert an    */
/*     iSeries file name "library/file(member)"  to a MS-DOS    */
/*     name with suffix .TXT to the member name only            */
/*     such as "member.TXT".                                    */
/*     7.  If the flag is *IFS, the program will format a       */
/*     file name "library/file(member)" formatted for the       */
/*     iSeries QSYS.LIB Integrated File System. for example     */
/*      "/QSYS.LIB/LIBRARY.LIB/FILE.FILE/MEMBER.MBR"            */
/*                                                              */
/*     8.  If the flag is SPLF1, the program will format a      */
/*     spool file name formated as:                             */
/*     "jobname/useriud/jobnumber/splfname/splfnbr.suffix"      */
/*     and rebuild a new name with data from the CVTDATA        */
/*     parameter.                                               */
/*                                                              */
/*                                                              */
/*NOTE: PKZIP and PKUNZIP performs no validity checking on the  */
/*  name that is passed back and assumes that the name is valid */
/*  and unique.  If the name is not valid or unique, open errors*/
/*  or missing files may occur.                                 */
/*                                                              */
/*                                                              */
/**************************************************************/
 CVTNAME:    PGM        PARM(&INNAME &OUTNAME &CVTFLAGS &CVTDATA)
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*    Parameter  Program Variables                              */
/*      Input    Name that is used examine for change          */
 INNAME:     DCL        VAR(&INNAME) TYPE(*CHAR) LEN(256)
/*      Changed name that is passed back to PKZIP for IBM i     */
 OUTNAME:    DCL        VAR(&OUTNAME) TYPE(*CHAR) LEN(256)
/*     Flags  from PKZIP for IBM i  for controlling logic flow  */
 CVTFLAGS:   DCL        VAR(&CVTFLAGS) TYPE(*CHAR) LEN(5)
/*     data   from PKZIP command for controlling logic flow     */
 CVTDATA:    DCL        VAR(&CVTDATA) TYPE(*CHAR) LEN(256)
/*                                                              */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*     Program Variables                                        */
/* iSeries QSYS file name represented like "libnm/filenm(mbrnm)" */
 LIBNM:      DCL        VAR(&LIBNM) TYPE(*CHAR) LEN(10) /* Library +
                          Name */
 FILENM:     DCL        VAR(&FILENM) TYPE(*CHAR) LEN(10) /* File +
                          name */
 MBRNM:      DCL        VAR(&MBRNM) TYPE(*CHAR) LEN(10) /* Member +
                          name */
             DCL        VAR(&IDX)    TYPE(*DEC)  LEN(3)
             DCL        VAR(&IDXPRV) TYPE(*DEC)  LEN(3)
             DCL        VAR(&LEN)    TYPE(*DEC)  LEN(3)
          /* DCL        VAR(&MSGVAR) TYPE(*CHAR) LEN(300)              */
          /* DCL        VAR(&MUVAR)  TYPE(*CHAR) LEN(6) VALUE('..OUT=') */
```

```
/*   Spool  file name represented like             */
/* "JOBNAM/USERID/JOBNBR/SPLFNAM/SPLFNBR.SPLSUFIX" */
 JOBNAM:     DCL        VAR(&JOBNAM) TYPE(*CHAR) LEN(10) /* Job +
                          Name     */
 USERID:     DCL        VAR(&USERID) TYPE(*CHAR) LEN(10) /* User +
                          Id   */
 JOBNBR:     DCL        VAR(&JOBNBR) TYPE(*CHAR) LEN(7) /* Job +
                          Number  */
 SPLFNAM:    DCL        VAR(&SPLFNAM) TYPE(*CHAR) LEN(10) /* Spool +
                          File name  */
 SPLFNBR:    DCL        VAR(&SPLFNBR) TYPE(*CHAR) LEN(5) /* Spool +
                          File Number  */
 SPLSUFIX:   DCL        VAR(&SPLSUFIX) TYPE(*CHAR) LEN(4) /* +
                          Suffix      */
/* blank out name uncase no hit */
             CHGVAR     VAR(&OUTNAME) VALUE('    ')
/*                                                          */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  Flag set to *400                                        */
             IF         COND(&CVTFLAGS *EQ '*400') THEN(DO)

             CHGVAR     VAR(&OUTNAME) VALUE(&INNAME)

             GOTO       CMDLBL(ENDCHG)
             ENDDO
/*                                                           */
/* * * * * * * * * * * * ** * * * * * * * * * * * * * * * * */
/*  Flag set to O4MS                                        */
/*  Change iSeries  Library/File(Member)  to                */
/*         MSDOS        library/File/Member                 */

 O4MS:       IF         COND((&CVTFLAGS *EQ 'O4MS') *OR (&CVTFLAGS +
                          *EQ '*MSD') *OR (&CVTFLAGS *EQ '*MSM') +
                          *OR (&CVTFLAGS *EQ '*MST')  +
                          *OR (&CVTFLAGS *EQ '*IFS')) +
                          THEN(DO)
             CHGVAR     VAR(&IDX)    VALUE(0)
             CHGVAR     VAR(&IDXPRV) VALUE(1)
/*    Find a Library     Note:max length must be 11  */
 DOLIB1:     CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             IF         COND(%SST(&INNAME &IDX 1) *NE '/')       +
                          THEN(GOTO CMDLBL(DOLIB1))
             CHGVAR     VAR(&LEN) VALUE(&IDX - &IDXPRV)
             IF         COND(&LEN > 10)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(10))
             CHGVAR     VAR(&LIBNM) VALUE(%SST(&INNAME &IDXPRV &LEN))
/*    Find a File        Note:max length must be 11  */
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX + 1)
 DOFILE1:    CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             IF         COND(%SST(&INNAME &IDX 1) *NE '(')       +
                          THEN(GOTO CMDLBL(DOFILE1))
             CHGVAR     VAR(&LEN) VALUE(&IDX - &IDXPRV)
             IF         COND(&LEN > 10)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(10))
             CHGVAR     VAR(&FILENM) VALUE(%SST(&INNAME &IDXPRV &LEN))
/*    Find a Member      Note:max length must be 11  */
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX + 1)
 DOMBR1:     CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             IF         COND(%SST(&INNAME &IDX 1) *NE ')')       +
                          THEN(GOTO CMDLBL(DOMBR1))
             CHGVAR     VAR(&LEN) VALUE(&IDX - &IDXPRV)
             IF         COND(&LEN > 10)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(10))
             CHGVAR     VAR(&MBRNM) VALUE(%SST(&INNAME &IDXPRV &LEN))
             CHGVAR     VAR(&OUTNAME) VALUE(&INNAME)
  /* Save the new name lib/file/mbr */
DOOUT1:
 AMSD:       IF (&CVTFLAGS *EQ '*MSD')         +
               DO
                   CHGVAR VAR(&OUTNAME) VALUE(&LIBNM *TCAT '/' +
```

```
                        *TCAT &FILENM *TCAT '.TXT ')
                GOTO       CMDLBL(ENDCHG)
             ENDDO
 AMSM:       IF (&CVTFLAGS *EQ '*MSM')         +
               DO
                  CHGVAR VAR(&OUTNAME) VALUE(&LIBNM *TCAT '/' +
                         *TCAT &FILENM *TCAT '/' *TCAT &MBRNM +
                         *TCAT '.TXT ')
                  GOTO       CMDLBL(ENDCHG)
             ENDDO
 AMST:       IF (&CVTFLAGS *EQ '*MST')         +
               DO
                  CHGVAR VAR(&OUTNAME) VALUE(&MBRNM +
                         *TCAT '.TXT ')
                  GOTO       CMDLBL(ENDCHG)
             ENDDO
 AIFS:       IF (&CVTFLAGS *EQ '*IFS')         +
               DO
                  CHGVAR VAR(&OUTNAME) VALUE('/QSYS.LIB/' *TCAT +
                         &LIBNM *TCAT '.LIB/' *TCAT &FILENM *TCAT +
                         '.FILE/' *TCAT &MBRNM *TCAT '.MBR')
                  GOTO       CMDLBL(ENDCHG)
             ENDDO

             CHGVAR     VAR(&OUTNAME) VALUE(&LIBNM *TCAT '/'      +
                        *TCAT &FILENM  *TCAT  '/'                 +
                        *TCAT &MBRNM   *TCAT ' ')
ENDO4MS:     GOTO       CMDLBL(ENDCHG)
          ENDDO    /* end of O4MS do flag */
/*                                                                 */
/* * * * * * * * * * * * ** * * * * * * * * * * * * * * * * * * * */
/*  Flag set to MSO4                                               */
/* Change a MSDOS file /dir1/dir2/../dirn/file.xxx  to             */
/*                       dir2/dirn(filexxx)  iSeries file          */

MSO4:     IF         COND(&CVTFLAGS *EQ 'MSO4') THEN(DO)
             CHGVAR     VAR(&IDX)    VALUE(0)
             CHGVAR     VAR(&IDXPRV) VALUE(1)
             CHGVAR     VAR(&LIBNM)  VALUE('          ')
             CHGVAR     VAR(&FILENM) VALUE('          ')
             CHGVAR     VAR(&MBRNM)  VALUE('          ')
/*    first find 1st blank and work backwards                      */
FINDLST2:    CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             if         COND(&IDX *GT 250) THEN(GOTO CMDLBL(ENDCHG))
             IF         COND(%SST(&INNAME &IDX 1) *NE ' ')       +
                          THEN(GOTO CMDLBL(FINDLST2))
/*    Find a Member      Note:max length must be 11  */
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX - 1)
 DOMBR2:     CHGVAR     VAR(&IDX) VALUE(&IDX - 1)
             IF         COND(&IDX *LT 1) THEN(GOTO CMDLBL(ENDCHG))

             IF         COND(%SST(&INNAME &IDX 1) *NE '/')       +
                          THEN(GOTO CMDLBL(DOMBR2))
             CHGVAR     VAR(&LEN) VALUE(&IDXPRV - &IDX)
             IF         COND(&LEN > 11)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(11))
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX - 1)
             CHGVAR     VAR(&MBRNM) VALUE(%SST(&INNAME &IDXPRV &LEN))
       /* REMOVE . IF ANY AND FORCE LENGTH TO 10  */

/*    Find a File        Note:max length must be 10  */
 DOFILE2:    CHGVAR     VAR(&IDX) VALUE(&IDX - 1)
             IF         COND(&IDX *LT 1) THEN(GOTO CMDLBL(SETNAME2))

             IF         COND(%SST(&INNAME &IDX 1) *NE '/')       +
                          THEN(GOTO CMDLBL(DOFILE2))
             CHGVAR     VAR(&LEN) VALUE(&IDXPRV - &IDX)
             IF         COND(&LEN > 11)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(11))
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX - 1)
             CHGVAR     VAR(&FILENM) VALUE(%SST(&INNAME &IDXPRV &LEN))
```

```
/*    Find a Library        Note:max length must be 10  */
 DOLIB2:     CHGVAR     VAR(&IDX) VALUE(&IDX - 1)
             IF         COND(&IDX *LT 1) THEN(GOTO CMDLBL(ENDCHG))

             IF         COND(%SST(&INNAME &IDX 1) *NE '/')       +
                          THEN(GOTO CMDLBL(DOLIB2))
             CHGVAR     VAR(&LEN) VALUE(&IDXPRV - &IDX)
             IF         COND(&LEN > 11)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(11))
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX - 1)
             CHGVAR     VAR(&LIBNM) VALUE(%SST(&INNAME &IDXPRV &LEN))

  /* Save the new name lib/file/mbr */
SETNAME2:
             IF     COND(&LIBNM *NE ' ') THEN(DO)
                      CHGVAR     VAR(&LIBNM) VALUE(&LIBNM *TCAT '/')
                      ENDDO
             IF     COND(&FILENM *NE ' ')  THEN(DO)
                       CHGVAR     VAR(&FILENM) VALUE(&FILENM *TCAT '/')
                       ENDDO
             CHGVAR     VAR(&OUTNAME) VALUE(&LIBNM *TCAT &FILENM      +
                        *TCAT &MBRNM   *TCAT ' ')
ENDMSO4:     GOTO       CMDLBL(ENDCHG)
          ENDDO    /* end of O4MS do flag */


/*                                                                   */
/* * * * * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * */
/*  Flag set to SPLF1  Sample                                        */
/* Change  "jobname/useriud/jobnumber/splfname/splfnbr.suffix"       */
/*      and build new name with 10 bytes of the CVTDATA field        */
/*      with a '.' separator followed by splfnbr.suffix              */
/*      for CVTDATA.splfnbr.suffix                                   */
/*  all fields from the Spool File passed file name are built        */
/*  care should be taken not to build duplicate file names in        */
/*  Archive                                                          */
/* * * * * * * * * * * * ** * * * * * * * * * * * * * * * * * * * * */


 SPLF1:       IF         COND((&CVTFLAGS *EQ 'SPLF1')) +
                          THEN(DO)
             CHGVAR     VAR(&IDX)    VALUE(0)
             CHGVAR     VAR(&IDXPRV) VALUE(1)
/*    Find a Jobname     Note:max length must be 10  */
 JOBNAM1:    CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             IF         COND(%SST(&INNAME &IDX 1) *NE '/') THEN(GOTO +
                          CMDLBL(JOBNAM1))
             CHGVAR     VAR(&LEN) VALUE(&IDX - &IDXPRV)
             IF         COND(&LEN > 10)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(10))
             CHGVAR     VAR(&JOBNAM) VALUE(%SST(&INNAME &IDXPRV &LEN))
/*    Find a User ID     Note:max length must be 10  */
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX + 1)
 USERID1:    CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             IF         COND(%SST(&INNAME &IDX 1) *NE '/')       +
                          THEN(GOTO CMDLBL(USERID1))
             CHGVAR     VAR(&LEN) VALUE(&IDX - &IDXPRV)
             IF         COND(&LEN > 10)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(10))
             CHGVAR     VAR(&USERID) VALUE(%SST(&INNAME &IDXPRV &LEN))
/*    Find a Job Number     Note:max length must be 6  */
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX + 1)
 JOBNBR1:    CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             IF         COND(%SST(&INNAME &IDX 1) *NE '/')       +
                          THEN(GOTO CMDLBL(JOBNBR1))
             CHGVAR     VAR(&LEN) VALUE(&IDX - &IDXPRV)
             IF         COND(&LEN > 7)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(7))
             CHGVAR     VAR(&JOBNBR) VALUE(%SST(&INNAME &IDXPRV &LEN))

/*    Find a SPLF Name      Note:max length must be 10  */
```

```
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX + 1)
 SPLFNAM1:   CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             IF         COND(%SST(&INNAME &IDX 1) *NE '/')       +
                          THEN(GOTO CMDLBL(SPLFNAM1))
             CHGVAR     VAR(&LEN) VALUE(&IDX - &IDXPRV)
             IF         COND(&LEN > 10)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(10))
             CHGVAR     VAR(&SPLFNAM) VALUE(%SST(&INNAME &IDXPRV &LEN))

/*    Find a SPLF nbr      Note:max length must be 5  */
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX + 1)
 SPLFNBR1:   CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             IF         COND(%SST(&INNAME &IDX 1) *NE '.')       +
                          THEN(GOTO CMDLBL(SPLFNBR1))
             CHGVAR     VAR(&LEN) VALUE(&IDX - &IDXPRV)
             IF         COND(&LEN > 5)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(5))
             CHGVAR     VAR(&SPLFNBR) VALUE(%SST(&INNAME &IDXPRV &LEN))

/*    Find a Suffix      Note:max length must be 4  */
             CHGVAR     VAR(&IDXPRV) VALUE(&IDX + 1)
 SPLSUFIX1:  CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             IF         COND(%SST(&INNAME &IDX 1) *NE ' ')       +
                          THEN(GOTO CMDLBL(SPLSUFIX1))
             CHGVAR     VAR(&LEN) VALUE(&IDX - &IDXPRV)
             IF         COND(&LEN > 4)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(4))
             CHGVAR     VAR(&SPLSUFIX) VALUE(%SST(&INNAME &IDXPRV &LEN))

/*    Set CVTDATA       Note:max length must be 4  */
             CHGVAR     VAR(&IDX) VALUE(0)
 CVTDATA1:   CHGVAR     VAR(&IDX) VALUE(&IDX + 1)
             IF         COND(%SST(&CVTDATA &IDX 1) *NE ' ') +
                          THEN(GOTO CMDLBL(CVTDATA1))
             CHGVAR     VAR(&LEN) VALUE(&IDX)
             IF         COND(&LEN > 10)                          +
                          THEN(CHGVAR VAR(&LEN) VALUE(10))
             CHGVAR     VAR(&LIBNM) VALUE(%SST(&CVTDATA 1 &LEN))
             IF         COND(&LEN > 1)                           +
                         THEN(CHGVAR VAR(%SST(&CVTDATA &LEN 1))  +
                          VALUE('.'))

  /* Save the new name lib/file/mbr */
             CHGVAR     VAR(&OUTNAME) VALUE(&LIBNM *TCAT &SPLFNBR +
                          *TCAT '.' *TCAT &SPLSUFIX *TCAT ' ')


ENDSPLF1:     GOTO       CMDLBL(ENDCHG)
          ENDDO    /* end of SPLF1 do flag */
/*                                                               */
ENDERR:      CHGVAR     VAR(&OUTNAME) VALUE('   ')
/*                                                               */
/* * * * * * * * * * * * ** * * * * * * * * * * * * * * * * * * */

/* End of CVTNAME for return                                     */
 ENDCHG:
        /*     CHGVAR     VAR(&MSGVAR) VALUE(&INNAME *BCAT &MUVAR + */
        /*                  *BCAT &OUTNAME)                        */
        /*   SNDPGMMSG  MSGID(AQZ0000) MSGF(*LIBL/PKZIPMSG)        */
        /*                MSGDTA(&MSGVAR) TOPGMQ(*PRV)             */
        /*   DMPCLPGM    CL dump of variable to help debug        */

 ENDEXIT:    ENDPGM
```

D Implementation Considerations

***Smartcrypt**i* components have been structured and written for the iSeries platforms and iSeries servers. The code base includes support for the IBM i operating system.

As with any product placed into production, familiarity with (and verification of) features is always highly recommended. Below is a list of key features that users of PKZIP should thoroughly understand.

Key Features

- The commands PKZIP and PKUNZIP contain all parameters and options required to use ***Smartcrypt**i*.** The parameters and options are in a format that provides a migration path toward the open systems environment. All parameters in the two commands are supported with the IBM i OS help panels. The parameters and options should be reviewed to understand the features supported.
- With the IFS, long file names are supported for files, archive files, and list files.
- List files for inputting file selections and outputting selections is supported for both the QSYS library file system and the IFS.
- Archive files are supported in both the QSYS library file system and the IFS. By using the archive files within the IFS, performance is enhanced. CPU operation is more efficient and archive files are smaller. As a result, elapsed run time is reduced.
- Provides a quick and easy method for installing on the iSeries without running under the QSECOFR user or security authority.
- Using the parameter MSGTYPE, messages can be directed to a printer file, the message queue, or both.
- When archive files are created in the QSYS library file system, the record length can be defined to fit the customer environment.
- When a file is extracted to the QSYS library file system, and neither the file nor the extended attribute for the record length exists, the default file's created record length can be defined externally.

- When compressing a file in text mode, you can define the record as a fixed-length record with trailing blanks that are based on the file's record length description in the QSYS library file system.
- When selecting files for compression in the IFS, you can specify whether to search recursively through the directories for file selection, or to only search the currently specified directory.
- When extracting files with the PKUNZIP command, you can drop the path of files in the archive and use only the file name. This allows you to build the extracted file in new directories, libraries, and/or files. This applies to IFS and QSYS library file systems.
- When using files with the product, the attributes are stored in the archive to indicate how the file was stored (binary, text, text in EBCDIC, SAVF, or Database Services) and can be used when extracting files with FILETYPE(*DETECT).
- The PKUNZIP command allows for defining default paths and libraries for both IFS and QSYS library file systems.
- File selection has been written to provide additional file filtering controls. Individual file selection can be made using the FILE parameter and/or the "List File" (INCLFILE) which allows a list of files to be selected.
- File exclusion has been written to provide additional file filtering controls. Individual file exclusion can be made using the EXCLUDE parameter and/or the "List File" (EXCLFILE), which allows a list of files to be excluded.
- ***Smartcrypt**i* is a licensed product, and without proper licensing, it can only be used to view archives. Licensing allows flexibility with the product features and hardware platforms. The license dataset must be defined and customized prior to use of the product.
- Extended attributes are stored within the archive directory.
- **Note:** These attributes are not published as part of a program control interface and may change between releases. Using the parameter TYPE(*VIEW) with VIEWOPT(*ALL) will report extended file information in the same report format order, regardless of the order that extended attributes are stored in the archive.
- ***Smartcrypt**i* library can be renamed from the standard distribution name of PKW14961S to fit the operational environment.
- PKZIP and PKUNZIP commands can be executed without the product's library being part of the library list.

E Creating Commands with New Defaults

There are times when some clients would like to change the defaults of the commands for their environment. This is why the source for the commands PKZIP, PKZSPOOL, and PKUNZIP are included in the distribution library in the QCMDSRC file. It is important that the sequences, values, and properties of the parameters do not change. To protect your ability to recover, it is suggested that you rename the previous command and command source and copy them before make any changes.

Another way to change the command defaults is to use the CHGCMDDFT command. Again, it is a good idea to capture your changes to the commands so they can be reapplied on any updates or upgrade. Some customers create a CLP in which to document the changes. They then just update defaults in the CLP when applying a PTF or upgrade.

For example maybe you want to change the PKZIP command in MYZIPLIB library to use the IFS file system as the default for all archives.

This command would be:

➔ **CHGCMDDFT CMD(MYZIPLIB/PKZIP) NEWDFT('TYPARCHFL(*IFS)')**

Or maybe there is a need to change all of the default translation tables for both PKZIP and PKUNZIP to translation file xxxxxxxxx.

➔ **CHGCMDDFT CMD(MYZIPLIB/PKZIP)**
NEWDFT('FTRAN(xxxxxxxxx) TRAN(xxxxxxxxx)')

➔ **CHGCMDDFT CMD(MYZIPLIB/PKUNZIP) NEWDFT('FTRAN(xxxxxxxxx)**
TRAN(xxxxxxxxx)')

If you want to restrict where the commands can be performed, then change the parameter ALLOW(*ALL) to the value you require. See IBM CL manuals or prompt CRTCM.

Below are sample statements to create the ***Smartcrypt***^{i} commands. In this example, the PKZIP library is PKW14961S.

Create PKZIP Command:

➔ **CRTCMD CMD(PKW14961S/PKZIP) PGM(PKW14961S/PKZIP)**
SRCFILE(PKW14961S/QCMDSRC)
HLPPNLGRP(PKW14961S/ ZIPHLP) HLPID(SZIPID)

Create PKZSPOOL Command

➔ **CRTCMD CMD(PKW14961S/PKZSPOOL) PGM(PKW14961S/PKZIP)**

SRCFILE(PKW14961S/QCMDSRC)
HLPPNLGRP(PKW14961S/ ZIPHLP) HLPID(SZIPID)

Create PKUNZIP Command:

➔ **CRTCMD CMD(PKW14961S/PKUNZIP) PGM(PKW14961S/PKUNZIP)**
SRCFILE(PKW14961S/QCMDSRC)
HLPPNLGRP(PKW14961S/ UZIPHLP) HLPID(SUZIPID)

F Extended Attributes

This chapter reviews the extended attributes that ***Smartcrypt***^{i} builds when using the parameters EXTRAFLD(*YES) or DBSERVICE(*YES). These extended attributes can be viewed for a file by using PKUNZIP with parameters TYPE(*VIEW) and VIEWOPT(*DETAIL) for the archive.

Within the ZIP archive are two directories that provide information about the files that are held within it. A local directory (included at the beginning of each file) contains information such as the file size and the date the file was zipped. The central directory (located at the end of the ZIP archive) lists the complete contents of the ZIP archive and is the primary source of information for UNZIP processing. In each directory there can exist extended data or attributes for the compressed files.

When EXTRAFLD(*YES) is specified, a set of basic attributes are created for each file in the archive. The type of attribute created depends on the type of file system (QSYS library file system or IFS) in which the file being compressed exists. The standard attributes for both systems are described below.

If the parameter DBSERVICE(*YES) is specified for the PKZIP command, and if the file being compressed is a physical file (PF-DTA or PF_SRC), then the basic extended attributes are created followed by the detailed database attributes. With the detailed database attributes, PKUNZIP can build and compile the DDS source file to recreate the database file (see DATABASE attributes). The database file will be compressed in the binary mode.

Standard QSYS Library File System Attributes

When the files being compressed are from the QSYS library file system, the standard attributes created are described in the following tables.

For additional information, see the DSPFD command in the IBM manuals.

Physical Files (Source and Data)

	Attribute	Description
1	Maximum Record Length	Specifies the length (in bytes) of the records stored in the physical file.
2	Library Text	The text description of the library that the file does exist.
3	File Text	The text description of the File.
4	Member Text	The text description of the member.
5	File Type	Specifies whether each member of the physical file being created contains data records or contains source records (statements. PF_SRC or PF_DTA.).
6	Source Type Code	Specifies the source type of the member, such as, CLP, PF, LF, RPGLE, etc.

SAVF

	Attribute	Description
1	Library Text	The text description of the library that the SAVF exists.
2	File Text	The text description of the SAVF.

Standard IFS Attributes

When the files being compressed are from the integrated file system, the standard attributes for stream files are as described in the following table.

	Attribute	Description
1	Code Page	
2	File creations date/time	The date and time when the file was created.
3	File last access date/time	The date and time when the file was last accessed.
4	File last modification date/time	The data and time when the file was last modified.

Advanced Encryption Attributes

When the files being compressed with advanced encryption, encryption attributes are added to the archive.

	Attribute	Description
1	Encryption Method/ Algorithm	Indicates the Encryption Method.
2	Encryption Key Type	The key type used with the encryption algorithm.
3	Security Method	Indicates whether a passphrase, certificate, or both passphrase and certificate are in use.

DATABASE Attributes

With the parameter DBSERVICE(*YES) in PKZIP, a special set of attributes are created for files that are physical files (both source and data types) in the QSYS

library file system. These attributes can be used by PKUNZIP to create a database by building the DDS source and issuing the CRTPF command for the source.

These database attributes are described in the following tables.

File Physical Attributes

For more information, see the CRTPF and CHGPF commands or IBM publications.

	Attribute	Description
1	File Record Format	The name of the file's record format.
2	File Record Text	The Text description for the file's format.
3	Maximum Number of Members	Specifies the maximum number of members that the physical file being created can have at any time.
4	Number Fields	The number of fields in the database.
5	Number of keys	The number of key fields in the database.
6	SIZE - nbr Records	SIZE parameter option - The number of records that can be inserted before an automatic extension occurs.
7	SIZE - increment value	SIZE parameter option - Value for the number of additional records.
8	SIZE - number of increments	SIZE parameter option - Maximum number of parts to be added automatically to the member.
9	Public Authority	AUT - Specifies the authority given to users who do not have specific authority to the physical file. **Note:** If an authorization list was specified for the file then AUT is set to *CHANGED. See "Database Attributes Considerations."
10	Record Description CCSID	The coded character set identifier for the record description.
11	Delete Percent	DLTPCT - Specifies the percentage of deleted records a file can contain before you want the system to send a message to the system history log.
12	Reuse deleted records	REUSEDLT - Specifies whether deleted record space should be reused on subsequent write operations.
13	Access path size	ACCPTHSIZ - Specifies the maximum size of auxiliary storage that can be occupied by access paths that are associated with keyed source physical files.
14	Access path maintenance	MAINT - Specifies the type of access path maintenance used for all members of the physical file.
15	Access path recovery	RECOVER - For files having immediate or delayed maintenance on their access paths, specifies when recovery processing of the file is done if a system failure occurs while the access path is being changed.

	Attribute	Description
16	Allocate storage	ALLOCATE - Specifies storage space is allocated for the initial number of records (SIZE parameter) for each physical file member when it is added.
17	Force keyed access path	FRCACCPTH - For files with keyed access paths only, specifies access path changes are forced to auxiliary storage along with the associated records in the file whenever the access path is changed.
18	Record format level check	LVLCHK - Specifies the record format level identifiers in the program are checked against those in the logical file when the file is opened.
19	Program describe File	Defines whether file is external file type.
20	Triggers Available	File had triggers defined. See “Database Attributes Considerations.”
21	File SQL Table	File is an SQL Table.
22	Constraints Available	File had Constraints defined. See “Database Attributes Considerations.”
23	File has Null Fields	File contains fields which allow NULL contents. See “Database Attributes Considerations.”

File Field Attributes

See “iSeries e DDS Reference Publication No. RBAF-P000-00” for a description of the keywords.

	Attribute	Description
1	Field Name	Field name.
2	Field Data Type	Specifies the data type associated with the field (such as, A-character, P-Packed, Decimal, etc.).
3	Field Length	Specifies the field length for named field.
4	Number of Decimals	Specifies the decimal placement within a zoned decimal field and the data type of the field as it appears in your program.
5	Field Usage	Specifies that a named field is an output-only or program-to-system field.
6	Field CCSID	(CCSID) - Specifies a coded character set identifier for character fields.
7	Field Text	(TEXT) - A text description (or comment) for the record format or field that is used for program documentation.
8	Column Headings	(COLHDG) - Specifies column headings used as a label for this field by various iSeries file tools.
9	Keyboard Shift Character	(REFSHIFT) - Specifies a keyboard shift for a field when the field is referred to in a display file or DFU operation.
10	Allow NULLS	(ALWNULL) - To define this field to allow the null value.
11	Variable Length	(VARLEN) - To define this field as a variable length field.

	Attribute	Description
12	Alias	(ALIAS) - Specifies an alternative name for the field.
13	Default Values	(DFT) - Specifies a default value for a field.
14	Edit Formatting values	(EDTCDE, EDTWRD, DATFMT, DATSEP, TIMFMT, TIMSEP) - Specifies editing for the field you are defining when the field is referenced later during display or printer file creation.
15	Validity Checking	(CHECK, COMP, RANGE, etc.) - Specifies validity checking in display files.
16	Indicator for Double Precision	An indicator for float type fields specifying single or double precision.
17	Allocated Length	The allocated length in bytes for data types that use variable lengths.

Key Field Attributes

See the IBM manual for the DDS description of keywords for KEYS.

	Attribute	Description
1	Key Field Name	The field name of the key.
2	Type of Sequence	Defines the type of key and the sequence of the key.

Database Attribute Considerations

Special database functionality and information such as triggers, file constraints, authorization lists, and alternate collating sequences are not stored in an archive.

1. If a file has fields that have the option ALWNULL, warning message AQZ0521 will be issued.
2. If the file contains triggers, warning message AQZ0518 will be issued. Any information about the triggers and associated programs are not stored in the archive.
3. If a file has file constraints, warning message AQZ0519 will be issued.
4. If an authorization list was specified for the file, then the AUT parameter is set to "*CHANGED." The authorization list is not stored in the archive. It is up to the user to set the authorization list when extracted.
5. If a file has an alternate collating sequence, warning message AQZ0520 will be issued. The information about the alternate collating sequence is not stored the archive.

If the database's triggers, file constraints, alternate collating sequence, and logical files are to be preserved, then save the database file, trigger programs (if they exist), and logical files to a SAVF. Then compress the SAVF.

Note: Using DBSERVICE(*YES) can increase the size of the archive because some databases may have hundreds of fields and attributes. If this archive will be used on a platform other than iSeries with ***Smartcrypt***^{i}, these attributes are ignored and therefore should not be used.

Source File Considerations

When compressing source files, the use of the archive file should be considered. By using the DBSERVICE(*NO), only the data is extracted in text mode. Other attributes such as the sequences numbers and change dates are not included. This would be the desired method if the source members are being transferred to another platform, or if the sequence number and changes dates are not important. This method would also give the best results for the total size of the archive.

If the sequence numbers and change dates are to be preserved, then you would use DBSERVICE(*YES) to store this source file as a database file. This would not work very well on non-EBCDIC platforms because the data is stored in binary mode with no EBCDIC to ASCII translation. The problem is that each member will have all available database attributes. This is still minimal because there are only three fields for a source database.

Spool File Attributes

When the files being compressed are from a spool file, the standard attributes are:

	Attribute	Description
1	Spool File Type	Device type SCS, AFP, etc.
2	Target Output Type	The type of file that was archived or converted (spool file, text, or PDF).
3	Internal Job and spool ID	Internal Job and Spool codes controlling the spool file.
4	Spool File description	The description of the file using the file name, file number, job, user and job number.
5	Pages	Number of pages in the spool file.
6	Code Page	The code page the was used to convert the spool file to TEXT or PDF.
7	OUTQ	The OUTQ and the OUTQ library the spool file resided.
8	User ID	The user who created the spool file

G Large File Support Licensing

Large File Support is a licensed feature in PKZIP for IBM i and as such will follow all other licensed features and consideration. In most cases care is taken at the beginning of each run to warn the customer of the ZIP64 requirements. One major difference is the ZIP64 feature could be activated during a large ZIP run where the archive crosses the boundary for ZIP64 support. In this situation the ZIP process will continue until the build has completed. A message will be issued and a temporary seven-day grace period will go into effect. Below is a summary list on what can activate the ZIP64 feature license.

- Any indication of ZIP64 in an existing Archive will require “Large Archive Support” to update the archive with PKZIP or extract with PKUNZIP.
- Large Archive Support includes:
- Encountering a ZIP64 End-Central configuration in an existing Archive.
- Central Directory Offset is beyond the 4 GB range.
- The number files in Archive exceeds 65,534.
- Encountering a Central Directory entry in ZIP64 format: (Uncompressed size, Compressed Size, or Local Offset exceeds 4 GB).
- When selecting files in PKZIP, the number of files in the old archive plus the new selected files exceeds 65,534 number of files.
- When selecting one or more files to compress and the file sizes exceed 4 GB.
- Trying to extract a file from an archive where the file size exceeds 4 GB.

Note: PKUNZIP will *VIEW and *TEST archives with ZIP64 condition without trying to check for Large System Support feature.

H History of Changes

Smartcrypt for IBM i RELEASE 16.0.0 B June 2017

New features available with ***Smartcrypt for IBM i*** Release 16.0.0 B include:

- 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

Smartcrypt for IBM i RELEASE 16.0.0 March 2016

New features available with ***Smartcrypt for IBM i*** Release 16.0.0 include:

- ***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.
- PKCFGSEC command changed with addition of the NSSRULES parameter.

- 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

SecureZIP for IBM i RELEASE 14.0.1 June 2014

New features available with ***SecureZIP for IBM i*** Release 14.0.1 include:

- All accumulated fixed issues
- Support OpenSSL 1.0.1e
- Support zlib and Bzip2 compression methods for 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.

SecureZIP for IBM i RELEASE 14.0 May 2012

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.

SecureZIP for i5/OS RELEASE 10.0.5 May 2010

New features include:

- All accumulated bug fixes.
- 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 i5/OS 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 latest Signed 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 inlist file parameters (INCLFILE/EXCLFILE) from 30 characters to 255 providing larger path lengths.
- Improved iPSRA exception handling for PKZIP and PKUNZIP.
- Updated ERROPT parameter for PKZIP, to assist skipping files with errors or iPSRA exceptions.
- PKZIP's parameter STOREPATH default was changed to be *REL in place of *YES.

PKZIP for i5/OS RELEASE 10.0 October 2007

New features include:

- All accumulated bug fixes
- Ability to perform passphrase-based decryption of SecureZIP archives and associated files
- Enhanced self-extraction support for strongly encrypted archives, and large archive support on specified releases of AIX, HP/UX, LINUX, Sun Solaris or Windows
- 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

SecureZIP for i5/OS RELEASE 10.0 October 2007

New features 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
- 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

PKZIP for i5/OS RELEASE 9.0 June 2006

New features include:

- All accumulated bug fixes
- 1Step2Tape feature added to create archives directly to tape
- Expanded maximum passphrase length from 200 to 260 alphanumeric characters
- Passphrase can be retrieved from a file utilizing the *INLIST control of passphrase
- SAVDLO added to iPSRA feature

SecureZIP for i5/OS RELEASE 9.0 June 2006

- All accumulated bug fixes
- 1Step2Tape feature added to create archives directly to tape
- Passphrase can be retrieved from a file utilizing the *INLIST control of passphrase
- Expanded maximum passphrase length from 200 to 260 alphanumeric characters

- Support Multiple Contingency Keys with the use of inlist for a type code
- SAVDLO added to iPSRA feature
- Renamed the PartnerLink SecureZIP for iSeries Reader/SecureLink program to “SecureZIP Partner for i5/OS.”

PKZIP for iSeries RELEASE 8.2 October 2005

- New compression algorithms with varying custom controls.
- Significant performances improvements with new compression algorithms.
- New ZIP64 signal constraint checks.
- New default internal translation tables for EBCDIC to ADCII.
- A separate input archive can be specified other than the archive file to created. This will allow an inputted archive to be preserved.
- A special key word *COPY for the FILES parameter has been added that allows a zip run that will only copy the files from another archive.
- The ability to extract zSeries files created with RDW (EBCDIC variable length records.
- More translations tables included for ASCII-EDCDIC translations
- Provided the ability to have embedded spaces in the EXCLUDE file names
- Added the new option *LIBS to Spool File selection parameter SFQUEUE. If *LIBS (Library Scan) for the OUTQ and a valid OUTQ library name are specified, PKZIP will only select spool files from all OUTQs in the specified library.
- Expanded the CRTLIST file name parameter to handle up to 255 characters for longer paths in the IFS
- Added *SIMULATE to the CRTLIST parameter in the PKZIP command. *SIMULATE will list all files selected instead of writing the selected list to a file.
- The ability to extract zSeries files created with RDW (EBCDIC variable length records.
- iSeries PKWARE Save/Restore Application feature or iPSRA.

SecureZIP for iSeries RELEASE 8.2 October 2005

- Added the PKWARE PartnerLink ***SecureZIP for iSeries Reader/SecureLink*** application to the SecureZIP product suite for the iSeries.
- New compression algorithms with varying custom controls.
- Significant performances improvements with new compression algorithms.
- New ZIP64 signal constraint checks.
- New default internal Translation Tables for EBCDIC to ADCII.

- A separate input archive can be specified other than the archive file to created. This will allow an inputted archive to be preserved.
- A special key word *COPY for the FILES parameter has been added that allows a zip run that will only copy the files from another archive.
- The ability to extract zSeries files created with RDW (EBCDIC variable length records.
- iSeries PKWARE Save/Restore Application feature or iPSRA.

SecureZIP for iSeries RELEASE 8.1 April 2005

- File and archive signing
- File and archive authentication
- Digital signing and authentication
- Support for a static certificate revocation list
- Contingency key for use with strong encryption
- Product separation: PKZIP for iSeries and SecureZIP for iSeries
- BSAFE® encryption
- More encryption algorithms for strong security
- Performance improvement with AES encryption
- ZIP and UNZIP supports encrypting and decrypting using digital certificates.
- Support for storing and managing digital certificates
- Support for utilizing directory integration (LDAP) for public certificate access
- File name encryption
- More translations tables included for ASCII-EDCDIC translations
- Provided the ability to have embedded spaces in the EXCLUDE file names
- Added the new option *LIBS to Spool File selection parameter SFQUEUE. By specifying *LIBS (Library Scan) for the OUTQ and a valid OUTQ library name, PKZIP will only select spool files from all OUTQs in the specified library.
- Expanded the CRTLIST file name parameter to handle up to 255 characters for longer paths in the IFS
- Added *SIMULATE to the CRTLIST parameter in the PKZIP command. *SIMULATE will list all files selected instead of writing the selected list to a file.
- Default Changes: For SecureZIP ADVCRYPT is now AES256
- New Commands:
- PKCFGSEC –Update Security Configuration
- PKLDAPTST -LDAP Testing

- PKBLDCDB –Build Certificate Locator Database
- PKSTORADM – Certificate Store Administration
- PKQRYCDB – Query Certificate Locator Database

Windows Compatibility

When using 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.

SecureZIP for i5/OS 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.

SecureZIP for i5/OS 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.

J

FIPS-197 Certification of PKZIP for AES

Advanced Encryption Standard FIPS Validation

PKZIP for OS/400 Version 5.5 and higher (which includes ***SecureZIP for i5/OS, SecureZIP for IBM i, and Smartcrypt for IBM i)*** use AES, which is the official US Government standard for encryption. The AES algorithm was approved as a Federal Information Processing Standard by the Commerce Department on May 26th, 2002.

The National Institute of Standards and Technology (NIST) has announced approval of the Federal Information Processing Standard (FIPS) for the AES algorithm (see NIST AES FIPS Information at http://csrc.nist.gov/CryptoToolkit/aes/).

The ***PKZIP for OS/400 Version 5.5*** implementation was validated in accordance with FIPS-197 for the Advanced Encryption Standard. Information regarding the validation specification and certifications of PKWARE products can be found at http://www.csrc.nist.gov/cryptval/aes/aesval.html ***(certificate #63)***.

The AES Algorithm

The Rijndael algorithm chosen uses a combination of advanced security, performance, efficiency, ease of implementation, and flexibility to make it the appropriate standard of advanced encryption for the AES.

Rijndael performs consistently in both hardware and software and in cross platform environments regardless of its use in feedback or non-feedback modes. Rijndael's key setup time is very good, and its key agility is excellent. Memory requirements are very low, making it the first choice for restricted-space environments, in which it also demonstrates high performance. Power and timing attacks are easily defended against due to Rijndael's operations.

Note that the AES was intentionally developed to replace DES.

AES Key Sizes

AES has three key sizes. They are: 128, 192, and 256 bits. Key sizes are depicted in the following decimal terms:

- 3.4×10^{38} possible 128-bit keys

- 6.2×10^{57} possible 192-bit keys
- 1.1×10^{77} possible 256-bit keys

In comparison, DES keys are only 56 bits, which means there are approximately 7.2×10^{16} possible DES keys. Therefore, there they are on the order of 10^{21} times more AES 128-bit keys than DES 56-bit keys.

K Contact Information

PKWARE, Inc.

Web Site: www.pkware.com

For licensing, contact Sales at 937-847-2374 or email PKSALES@PKWARE.COM.

For technical support, contact the Product Services Division at 937-847-2687 or visit the Support Web site: http://www.pkware.com/support/system-i.

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 then follow the steps outlined above for PKZIP for IBM i.

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-306.ibm.com/ibm/terminology

Absolute Path Name

A string of characters that is 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 stand 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 the user 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.

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 is 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.

License Authorization Code (LAC)

The inserted code that is needed to unlock an IBM i licensed program.

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 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.

QSYS

The library shipped with the iSeries, i5/OS and 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.

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 is 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.

Variable-Length

A characteristic of a file in which the individual records (and/or the file itself) can be of varying length. See also Fixed-Length.

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