



How to use Encryption Facility for z/VSE

Encryption Facility for z/VSE V1.1
Encryption Facility for z/OS V1.1
z/OS Java Client

Last formatted on: Friday, November 19, 2010

Joerg Schmidbauer
jschmidb@de.ibm.com

Dept. 3229
VSE Development
IBM Lab Böblingen
Schönaicherstr. 220

D-71032 Böblingen
Germany



Disclaimer

This publication is intended to help VSE system programmers setting up infrastructure for their operating environment. The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk. Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment. Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.
Microsoft, Windows, Windows XP, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

Contents

1	Introduction	4
2	Encryption Facility for z/VSE	4
2.1	Prerequisites	5
2.2	Corrective service.....	5
2.3	Relationship to Encryption Facility for z/OS	5
2.4	Obtaining the z/OS Java Client	6
2.5	Relationship to the TS1120 tape drive	6
3	Performance considerations.....	7
4	Password-based encryption	8
4.1	Encrypting on VSE.....	8
4.2	Decrypting on VSE	10
5	Public-key encryption.....	10
5.1	Encrypting on VSE.....	11
5.1.1	Creating an RSA key pair using the keytool.exe	11
5.1.2	Exporting the public key from the Java keystore.....	11
5.1.3	Import public key certificate into Keyman/VSE.....	12
5.1.4	Upload public key certificate to VSE keyring library	13
5.1.5	Do the encryption on VSE	14
5.1.6	Decrypting locally	14
5.1.7	Encrypting for multiple recipients	15
5.2	Decrypting on VSE	15
5.2.1	Create RSA key pair using Keyman/VSE.....	16
5.2.2	Upload the private key to VSE	16
5.2.3	Export the public key into a Java keystore	17
5.2.4	Encrypt locally	19
5.2.5	Do the decryption on VSE	19
6	Using record-based files	20
6.1	Encrypting on z/OS	20
6.2	Decrypting on VSE	21
6.3	Encrypting on VSE.....	21
6.4	Decrypting on z/OS.....	22
6.5	EBCDIC to ASCII considerations.....	22
7	Known problems.....	22
7.1	Misaligned records after decryption on VSE	22
7.2	Using CA DynamT.....	23
8	More information.....	23

Changes

Nov 12, 2008 – initial version.

Dec 2008 – some minor textual changes

Apr 2009 – added new section “Corrective service”

June 2010 – update on “corrective service” and new sections “Using record-based files” and “Known problems”

July 2010 – new section “Using CA DynamT” on page 23

1 Introduction

This paper describes the use of Encryption Facility for z/VSE V1.1 and how to exchange encrypted data with Encryption Facility for z/OS V1.1 and the z/OS Java Client.

The following software has been used in the test setup.

- z/VSE 4.2.0
- TCP/IP for VSE/ESA 1.5F with ZP15F230
- VSE Connector Server as part of z/VSE 4.2.0 (job STARTVCS)
- Java 1.6.0_05 from Sun Microsystems
- keytool.exe as part of Java 1.6.0
- Keyman/VSE (build date Oct 2007)
- Encryption Facility for z/VSE V1.1
- IBM Java Client as part of IBM Encryption Facility for z/OS Client V1.2

2 Encryption Facility for z/VSE

Encryption Facility for z/VSE, program number 5686-CF8-40, is an optional priced feature and provides data protection by offering the encryption of data for exchange, archiving, and backup purposes. It is a software-based tool and eligible for MWLC pricing. Depending on the kind of processor and the type of cryptographic hardware that you have installed, the Encryption Facility for z/VSE uses hardware-accelerated crypto support for encryption and decryption.

Supported file formats include single SAM files, VSAM files or VSE Library members, but also complete backups made with any backup tool either from IBM or vendors. For single VSAM files or VSE Library members the filenames are specified directly in the JCL invoking the tool. For full backups, you would first backup your data using any available backup tool to a real tape or virtual tape, and in a second step encrypt this backup tape to an encrypted dataset, which can be written to a second real tape, virtual tape, or DASD.

Encryption Facility for z/VSE makes use of triple-DES (TDES) and AES-128 algorithms for data encryption. On a system with TCP/IP for VSE/ESA, you can use Encryption Facility for z/VSE to generate TDES and AES keys and encrypt them for protection through RSA public keys of lengths 512, 1024, and 2048, but also using passwords to generate the encryption key. Password-based key generation is an option as well. RSA keys with key lengths of 2048 bit require a PCIXCC, Crypto Express2, or Crypto Express3 card.

On systems without TCP/IP for VSE/ESA (or equivalent), you can only use passwords for the generation of clear TDES and AES keys.

Documentation about Encryption Facility for z/VSE V1.1 is contained in the z/VSE V4.1 and V4.2 Administration Guide, available online at

<http://www.ibm.com/servers/eserver/zseries/zvse/documentation/#vse>

For commands and options refer to the Administration Guide.

2.1 Prerequisites

To use Encryption Facility for z/VSE V1.1 you must have

- Activated the CPU Assist Facility (CPACF, feature code #3863), which is available on z890, z990, z9, and z10 processors
- TCP/IP for VSE/ESA 1.5E or higher for public key encryption
- TCP/IP fix ZP15F230 for 2048 bit RSA keys
- A Crypto Express2 or equivalent for processing 2048 bit RSA keys
- At least 8 MB partition size due to the fact that the tool is an LE/VSE application.

Recommended fixes:

- TCP/IP fix ZP15F246, which solves the problem of getting a program check when trying to read a public key from a CERT member which does not exist

2.2 Corrective service

The following APARs and PTFs are currently available for Encryption Facility for z/VSE V1.1 and V1.2.

APAR	PTF	Date	Description
DY47051	UD53499	April 2009	Fixed problem "ERROR: NOT ALL DATA HAS BEEN PROCESSED" when encrypting/decrypting a CLRTAPE.
DY47097 DY47098	UD53550 UD53551	Feb 2010	Encryption Facility: problems fixed: IJBEO (DY47097 and DY47098): - LE return code handling IJBEPGP (DY47098 only): - handle rc < 0 for rec_read and strm_read - don't initialize/use ZLIB library when compression level = 0 - return with correct rc from main() - use info from RECORDINFO packet when opening the output file for decryption. LE APAR PM06694 fixes the problem of a program check when using an invalid value for LRECL.

2.3 Relationship to Encryption Facility for z/OS

Encryption Facility for z/VSE V1.1 has a close relationship to Encryption Facility for z/OS V1.1. It uses the same encrypted data format (System z format) as the z/OS based tool and is therefore able to exchange encrypted data with any z/OS system having Encryption Facility for z/OS installed.

Encryption Facility for z/OS consists of several parts, including two Web downloadable tools, the Java Client and the Decryption Client for z/OS. Both tools can be downloaded for free. The Java Client is intended for exchanging encrypted data with non-z platforms, while the Decryption Client is intended for decrypting data on z/OS systems where the full Encryption Facility product is not installed.

These two Web downloadable tools can also be used in a z/VSE environment, allowing for data exchange with non z/VSE platforms.

Note on software support: To report problems with the z/OS Java Client or z/OS Decryption Client when used to exchange data with the Encryption Facility for z/VSE, please contact: zvse@de.ibm.com. Software support for Encryption Facility for z/VSE is provided via the normal L1, L2, and L3 service.

Figure 1 shows the involved parts of Encryption Facility for z/OS and their relationship to Encryption Facility for z/VSE.

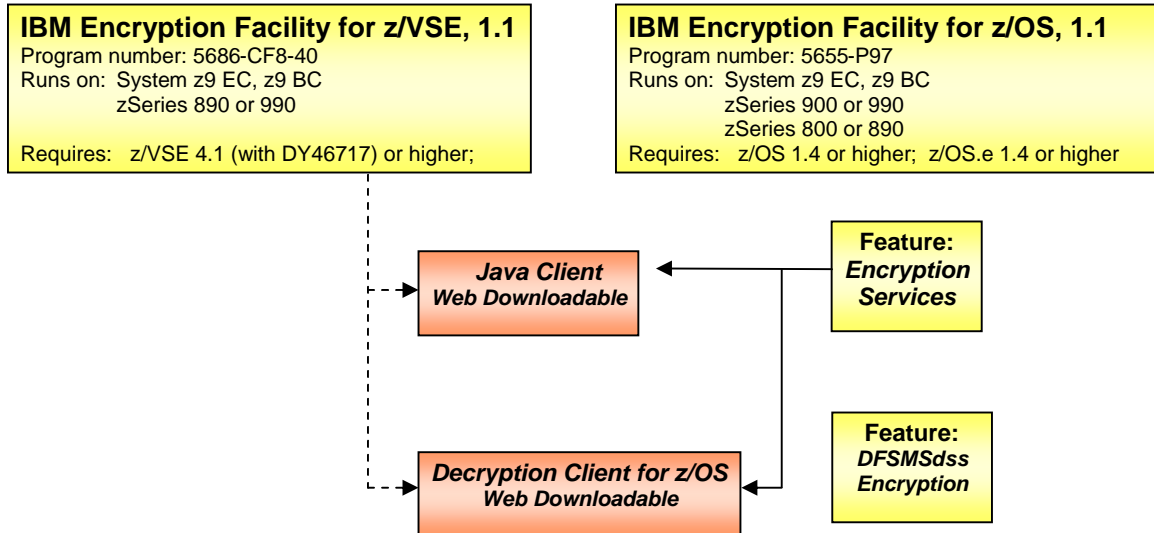


Figure 1: relationship of Encryption Facility for z/VSE and Encryption Facility for z/OS

2.4 Obtaining the z/OS Java Client

As described in the z/VSE V4.1 Administration Guide, you can download the z/OS Java Client from

<http://www.ibm.com/servers/eserver/zseries/zos/downloads/#efclient>

Unzip the downloaded zip-file in a new empty directory. Further installation is not necessary.

The next section describes how Encryption Facility is related to the TS1120 tape drive with encryption support.

2.5 Relationship to the TS1120 tape drive

The IBM TS1120 tape drive with encryption capability is supported by z/VSE V3.1 and later. Encryption is done by the tape drive itself, while key management is done via the so called “Encryption Key Manager” (EKM), a Java application that acts as a key server. The TS1120 uses public-key encryption and the related public keys are maintained by the EKM. For more details about the TS1120 support, refer to the z/VSE Administration book.

Encryption Facility for z/VSE complements the support for the TS1120 encrypting tape drives. While the TS1120 is the preferred solution for high volume backup/archive applications, Encryption Facility for

z/VSE is designed to allow secure exchange of encrypted data with other locations within your company, business partners, suppliers, and customers.

Password-based encryption is a simple but secure way for exchanging encrypted data with other locations, but also for creating local backup archives. The ability to write encrypted data to disk for further file transfers also complements the TS1120 solution.

Table 1 shows a list of various encryption scenarios and the preferred use of either the TS1120 or Encryption Facility.

	TS1120	Encryption Facility
High volume backup/archiving	x	-
Data encryption for rest on VSE disks	-	x
Data encryption for subsequent file transfer (e.g. FTP)	-	x
Local archiving	x	x
Data exchange with remote sites having TS1120	x	-
Use existing TS1120 environment	x	-
Data exchange with Encryption Facility for z/OS V1.1	-	x
Data exchange with non-z platforms (EF Java client)	-	x
Password-based encryption	-	x
Public key based encryption	x	x
Offload CPU cycles	x	-
Perform encryption on z/VSE V3.1	x	-
Perform encryption on z/VSE V4.1 or later	x	x
Perform encryption with a z800 or z900	x	-
Perform encryption on a z890, z990, z9, z10	x	x

Table 1: positioning of TS1120 to Encryption Facility

The following chapter lists some performance considerations regarding algorithms, compression, and hardware support.

3 Performance considerations

Overall performance of an encryption or decryption process depends on several parameters.

- Compression.** Compressing data prior to encryption usually speeds up the process, because less data has to be encrypted. Compression is performed via the System z provided hardware compression feature. You must not use compression when exchanging data with a workstation platform!
- Encryption algorithm.** There are significant differences in terms of speed between the supported encryption algorithms. AES usually performs much faster than TDES. When using public key encryption, there is no big difference between the different public key sizes, because only the data key is encrypted using a public key.
- Hardware support.** For example, you can use EF for VSE on a z890, because the required hardware feature CPACF is available on this machine. However, on a z890 only TDES is

supported by CPACF, while AES is not available.

- **Physical I/O.** When e.g. encrypting a KSDS file with many small records, the encrypted dataset usually has a much bigger record length, using the maximum possible of the underlying ESDS file. Therefore, writing encrypted data to disk requires much less I/O than later writing the decrypted dataset with its many small records during the decryption process.

The following chapter describes how to encrypt data on VSE using password-based encryption, transfer the encrypted dataset to a workstation, and finally decrypt the data using the z/OS Java client.

4 Password-based encryption

Password-based encryption does not require any key stores. The encryption key is directly derived from the password. Encryption Facility for z/VSE always converts the EBCDIC password specified by JCL to ASCII. But passwords are case sensitive, so make sure to specify your password correctly both in JCL and on any other related platform.

Do not use any NLS specific characters (e.g. German umlauts) in a password. This could cause problems when translating the password to ASCII depending on the used code page. Encryption Facility uses the following code pages by default:

- ASCII code page : **IBM-850**
- EBCDIC code page : **IBM-1047**

You can change the codepage using parameters ASCII_CODEPAGE and EBCDIC_CODEPAGE. You should specify the code page parameters **before** the PASSWORD parameter in order to have your code page active when translating the password.

To manage your passwords you may use any available tool from vendors, freeware, or shareware. In my opinion, the open source tool KeePass (<http://keepass.sourceforge.net>) is a good solution for maintaining passwords.

As with password-based encryption we don't need any special setup, we can directly start encrypting and decrypting files.

4.1 *Encrypting on VSE*

Following JCL encrypts a VSE library member using password-based encryption (PBE). The encrypted dataset is a VSAM ESDS cluster.


```

* $$ JOB JNM=ENCRYPT,CLASS=0,DISP=D
// JOB ENCRYPT VSE LIBRARY MEMBER
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.PROD,PRD2.DBASE)
// EXEC IJBEFVSE
ENCRYPT
DESC=' ENCRYPTION TEST'
CLRTDES
PASSWORD=MYPASSWD
COMPRESSION=NO
ICOUNT=234
CLRFILE=DD:PRD2.CONFIG(IPINIT00.L)
ENCFILE=DD:ENCDATA
/*
/&
* $$ EOJ

```

Notes:

- Don't use another VSE library member as output dataset, because its last record would get padded to 80 characters. Because of this change, it would be impossible to decrypt the file on a workstation.
- Don't use compression, because it's not possible to decompress data on a workstation.

Now download the encrypted dataset to your Windows PC.

```

ftp> get encdata ipinit00.enc
200 Command okay
150-About to open data connection
File:VSE.EF.ENCDATA
Type:Binary Recfm:FB Lrecl: 80 Blksize: 80
CC=ON UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON NAT=NO CONT=OFF
MODE=Stream STRU=File
150 File status okay; about to open data connection
226-Bytes sent: 14,496
Records sent: 1
Transfer Seconds: .20 ( 71K per second)
File I/O Seconds: .00 ( 14496-bytes)
226 Closing data connection
ftp: 14496 bytes received in 1.20Seconds 12.05Kbytes/sec.

```

The next step is decrypting the downloaded file using the z/OS Java Client. Open a command prompt and got to the directory where the Java Client is installed. The following command string must be entered in one single line. Multiple lines are used here for better reading.

```

java -Djava.encryption.facility.debuglevel=1
com.ibm.encryptionfacility.EncryptionFacility
-mode decrypt
-password MYPASSWD
-outputFile ipinit00.l
-inputFile ipinit00.enc

```

Note: on VSE the password is always specified in uppercase letters, so you have to use uppercase letters too when decrypting the file on a workstation.

As the decrypted file now contains EBCDIC characters, you have to upload the file e.g. to VM in order to view it correctly. This step would not be necessary when encrypting binary data on VSE or when the decrypted data are not intended to be human readable.

4.2 Decrypting on VSE

Now let's encrypt a local file using the z/OS Java Client, upload it to VSE, and finally decrypt it on VSE. Enter the password in uppercase letters.

```
java -Djava.encryption.facility.debuglevel=1
com.ibm.encryptionfacility.EncryptionFacility
-mode encrypt
-underlyingKey PBEWithSHA1And3DES
-password MYPASSWD
-inputFile mypic.jpg
-outputFile mypic.enc
-iterations 123
```

Now upload the encrypted file to VSE.

```
ftp> put mypic.enc encdata
200 Command okay
150>About to open data connection
  File:VSE.EF.ENCDATA
  Type:Binary Recfm:FB Lrecl:   80 Blksize:   80
  CC=ON UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON NAT=NO CONT=OFF
  MODE=Stream STRU=File
150 File status okay; about to open data connection
226-Bytes received: 11,576
  Records received: 145
  Transfer Seconds:   .03 ( 377K per second)
  File I/O Seconds:  .01 ( 1130K per second)
226 Closing data connection
ftp: 11576 bytes sent in 0.00Seconds 11576000.00Kbytes/sec.
```

The following JCL decrypts the encrypted dataset and writes the clear data into the clear dataset, which is also a VSAM ESDS cluster.

```
* $$ JOB JNM=DECRYPT,DISP=D,CLASS=0
// JOB DECRYPT
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.PROD,PRD2.DBASE)
// EXEC IJBEFVSE
DECRYPT
PASSWORD=MYPASSWD
CLRFIL=DD:CLRDATA
ENCFIL=DD:ENCDATA
/*
/&
* $$ EOJ
```

Check the job output for any error messages.

5 Public-key encryption

Public-key encryption requires the setup of key stores on both platforms. The encrypting site needs a public key while the decrypting site needs the corresponding private key.

The z/OS Java Client requires a Java keystore containing any keys to be used. Keyman/VSE (build date Oct 2007 or later) supports Java keystores (JKS files), however, an alternative for creating a Java keystore is using the `keytool.exe`, which is part of your Java installation and is located in the `jre/bin` directory. The following sections describe how to setup the Java keystore using the `keytool.exe`, while sections 5.2.1 to 5.2.3 describe the steps in Keyman/VSE.

5.1 *Encrypting on VSE*

When encrypting on VSE, we start on the PC side to create an RSA key pair in a Java keystore. Further tasks will involve:

- Export the public key from the Java keystore
- Import the public key certificate into Keyman/VSE
- Upload the certificate to VSE, where it can be accessed by Encryption Facility.

5.1.1 Creating an RSA key pair using the `keytool.exe`

In a Java keystore, RSA key pairs are always wrapped into a certificate. The following command string creates a private key certificate.

```
keytool -genkey -keyalg "RSA" -alias mykey -keypass mypasswd -keystore
efvse.jks -storepass mypasswd
```

It is important to specify key algorithm “RSA”, because the default is “DSA”, which is not supported on VSE. You have to specify some personal information to be used when creating the private certificate.

```
What is your first and last name?
 [Unknown]: Joerg Schmidbauer
What is the name of your organizational unit?
 [Unknown]: VSE Development
What is the name of your organization?
 [Unknown]: IBM
What is the name of your City or Locality?
 [Unknown]: Boeblingen
What is the name of your State or Province?
 [Unknown]: BW
What is the two-letter country code for this unit?
 [Unknown]: DE
Is CN=Joerg Schmidbauer, OU=VSE Development, O=IBM, L=Boeblingen, ST=BW, C=DE
co
rrect?
 [no]: y
```

The JKS file is now created. You can get a list of the `keytool` parameters at

<http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html>

5.1.2 Exporting the public key from the Java keystore

Now use the `keytool list` command to show the contents of the keystore.

```
keytool -list -keystore efvse.jks -storepass mypasswd
```

This command will produce output similar to the following:

```
Keystore type: JKS
Keystore provider: SUN
```

Your keystore contains 1 entry

```
mykey, Oct 29, 2008, PrivateKeyEntry,
Certificate fingerprint (MD5): 95:20:54:AC:3C:BF:96:42:1D:CF:E6:9E:7E:45:29:D5
```

Now export the certificate into a binary file:

```
keytool -export -alias mykey -file efvse.crt -keystore efvse.jks -storepass
mypasswd
```

You should get a message like:

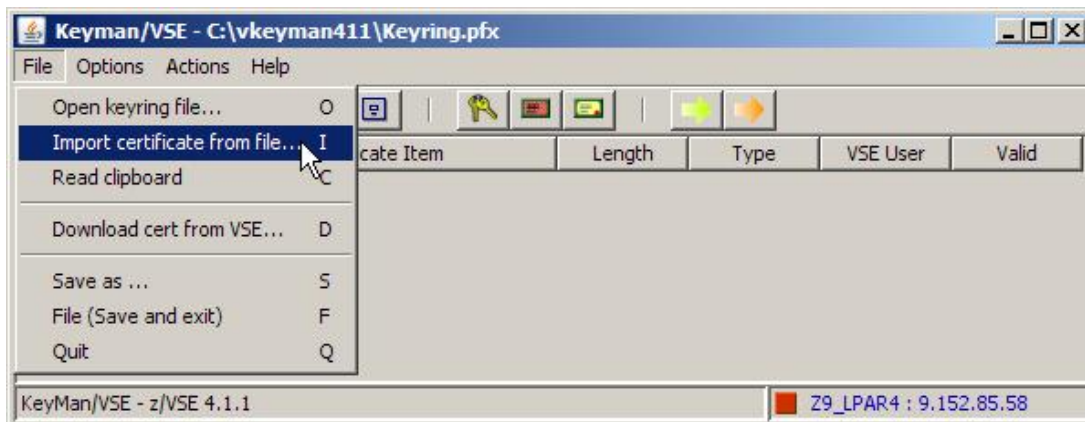
```
Certificate stored in file <efvse.crt>
```

Note that only the public key has been written into the output file. Private keys by definition never leave their original keystore.

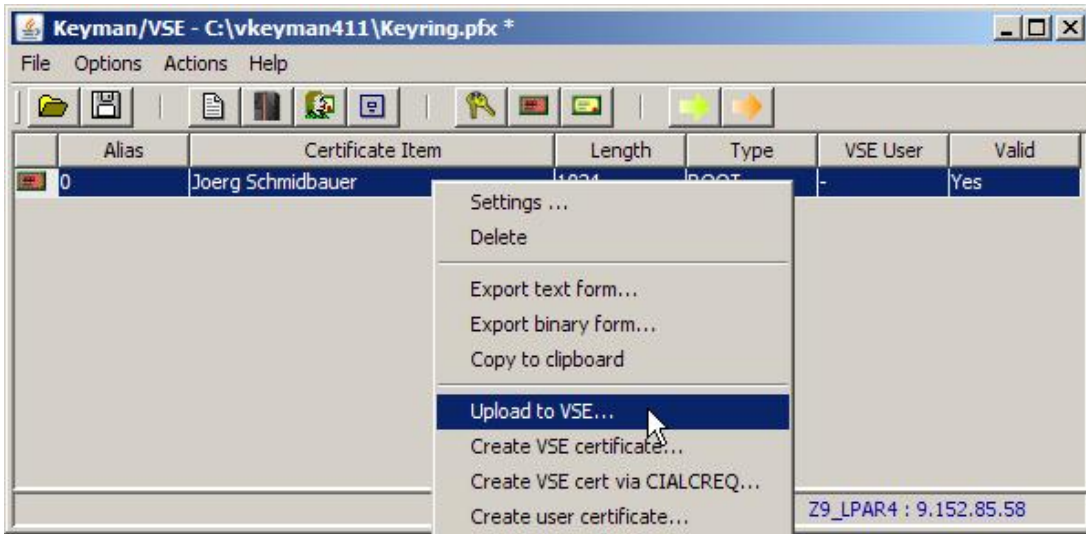
5.1.3 Import public key certificate into Keyman/VSE

If you did not already define your VSE system in Keyman/VSE, you should do that now. Refer to the Keyman/VSE “General help” section and click on links **How to - Basic VSE Host settings**.

To import the public key from the previously created file, select **File - Import certificate from file**.



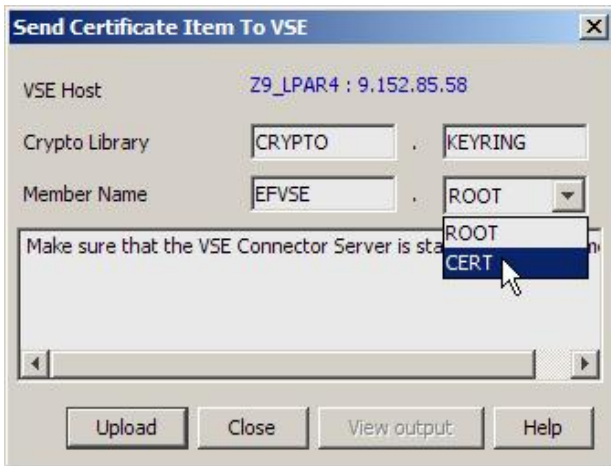
Navigate to the directory where the efvse.crt file is located and open the file. The certificate is now displayed in the Keyman main window. It is displayed as a ROOT certificate, because it is self-signed.



Now right-click the certificate and select **Upload to VSE**.

5.1.4 Upload public key certificate to VSE keyring library

Enter a unique member name and select CERT as the member type.



Press **Upload**.

The public key is now stored in the crypto library.

```
LD Efvse.*
```

```
DIRECTORY DISPLAY      SUBLIBRARY=CRYPTO.KEYRING      DATE: 2008-10-29
                                                                TIME: 20:28
-----
 M E M B E R      CREATION   LAST      BYTES   LIBR CONT SVA  A- R-
NAME      TYPE      DATE      UPDATE  RECORDS  BLKS  STOR  ELIG  MODE
-----
EFVSE     CERT      08-10-29  - -     609 B    1 YES  - - -
L113I RETURN CODE OF LISTDIR IS 0
```

5.1.5 Do the encryption on VSE

Prerequisites:

- The public key used for encryption must be now located on VSE in a CERT member.
- The corresponding private key must be contained in the Java keystore on the PC side.

In this example we will just put a JPG image into the clear dataset before running the following job.

```
* $$ JOB JNM=ENCRSA,CLASS=4,DISP=D
// JOB ENCRSA ENCRYPT USING PKE
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.PROD,PRD2.DBASE)
// EXEC IJBEFVSE
ENCRYPT
DESC='ENCRYPTION TEST'
CLRAES128
RSA=CRYPTO.KEYRING(EFVSE)
CLRFILE=DD:CLRDATA
ENCFILE=DD:ENCDATA
/*
/&
* $$ EOJ
```

Now check the job output. As Encryption Facility first tries to open a PRVK member with the given member name, there will be some LIBR errors. Finally, the public key from the CERT member should be taken.

```
T037: SSL303E IPDSCRFI failed RC=00000008(LIBROPIF) reason=00000418 00000008
T037: SSL113W IPDSCRFI get for CRYPTO KEYRING EFVSE PRVK failed
T037: SSL303E IPDSCRFI failed RC=000007E7(LIBRCALL) reason=000005D0
INFO: USING RSA PUBLIC KEY FROM CERTIFICATE:
      CRYPTO.KEYRING(EFVSE) (1024 BIT)
```

5.1.6 Decrypting locally

Now download the encrypted dataset in binary to your workstation and decrypt the file with the z/OS Java Client using the private key in the Java keystore.

```
ftp> get file2 mypic.enc
200 Command okay
150-About to open data connection
  File:VSE.VTAPE.FILE2
  Type:Binary Recfm:FB Lrecl:   80 Blksize:   80
  CC=ON UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON NAT=NO CONT=OFF
  MODE=Stream STRU=File
150 File status okay; about to open data connection
226-Bytes sent: 12,432
  Records sent: 1
  Transfer Seconds:   .00 ( 12432-bytes)
  File I/O Seconds:  .00 ( 12432-bytes)
226 Closing data connection
ftp: 12432 bytes received in 1.00Seconds 12.43Kbytes/sec.
```

The following command string will do the decryption.

```

java -Djava.encryption.facility.debuglevel=1
com.ibm.encryptionfacility.EncryptionFacility
-mode decrypt
-keyStoreName efvse.jks
-keyStoreType JKS
-keyStoreCertificateAlias mykey
-password mypasswd
-inputFile mypic.enc
-outputFile mypic2.jpg

```

Note that the password here specifies the keyring file password and must not be confused with the password as specified for password-based encryption.

5.1.7 Encrypting for multiple recipients

You can specify multiple RSA statements in the same job to allow multiple recipients to decrypt the encrypted dataset. This requires having the public key of each recipient available on VSE in a PRVK or CERT member.

```

* $$ JOB JNM=ENCMULT,CLASS=4,DISP=D
// JOB ENCMULT ENCRYPT WITH MULTIPLE PUBLIC KEYS
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.PROD,PRD2.DBASE)
// EXEC IJBEFVSE
ENCRYPT
DESC='ENCRYPTION TEST'
CLRAES128
RSA=CRYPTO.KEYRING(PUBKEY1)          <- refers to member PUBKEY1.CERT
RSA=CRYPTO.KEYRING(PUBKEY2)
RSA=CRYPTO.KEYRING(PUBKEY3)
RSA=CRYPTO.KEYRING(MYKEY)           <- refers to member MYKEY.PRVK
CLRFILE=DD:PRD2.CONFIG(IPINIT00.L)
ENCFILE=DD:ENCDATA
/*
/&
* $$ EOJ

```

In this example the data key is encrypted with the public keys of three CERT members and one PRVK member. This allows decrypting the output dataset on three remote systems where the corresponding private RSA keys are present. The remote systems can be z/OS or z/VSE systems, but also any Java workstation. In addition to this, the encrypted dataset can be decrypted on the same system using the public key that belongs to the private key specified in the last RSA control statement.

Note: TCP/IP fix **ZP15F246** is recommended to avoid a program check, which occurs when trying to access a CERT member which does not exist.

The maximum number of RSA statements is 16.

5.2 Decrypting on VSE

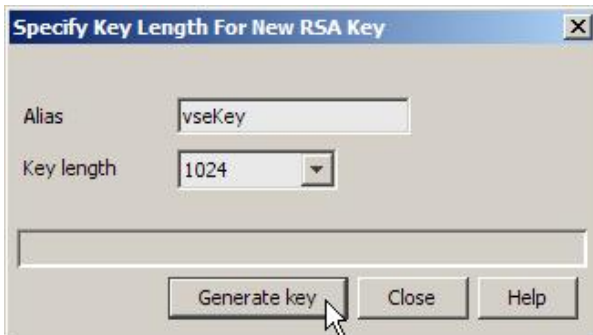
When decrypting on VSE, we need a PRVK member containing a private key. In this case the setup of our key store is different, as we start on the VSE side and export the VSE public key to the Java keystore on the workstation.

5.2.1 Create RSA key pair using Keyman/VSE

Open the Keyman/VSE tool and create a new RSA key pair.



Press the **Generate new RSA key pair** toolbar button.



Select the RSA key length and press **Generate key**.

5.2.2 Upload the private key to VSE

Start the VSE Connector Server on VSE in non-SSL mode and upload the key pair to VSE.



As this key is used for decryption with Encryption Facility, let's name the library member EFDECR.PRVK.



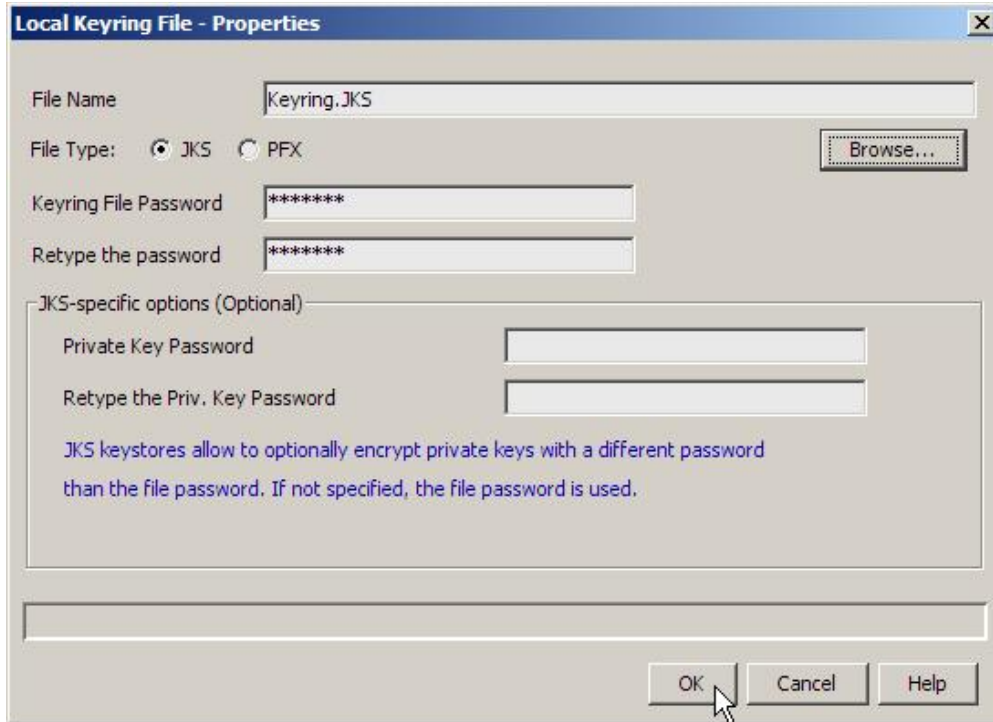
Press **Upload**. The key pair will now be stored in library member EFDECR.PRVK in the specified keyring library. The next step is to export the public key into a Java keystore where it can be used by the z/OS Java Client.

5.2.3 Export the public key into a Java keystore

Press the toolbar button **Save**.



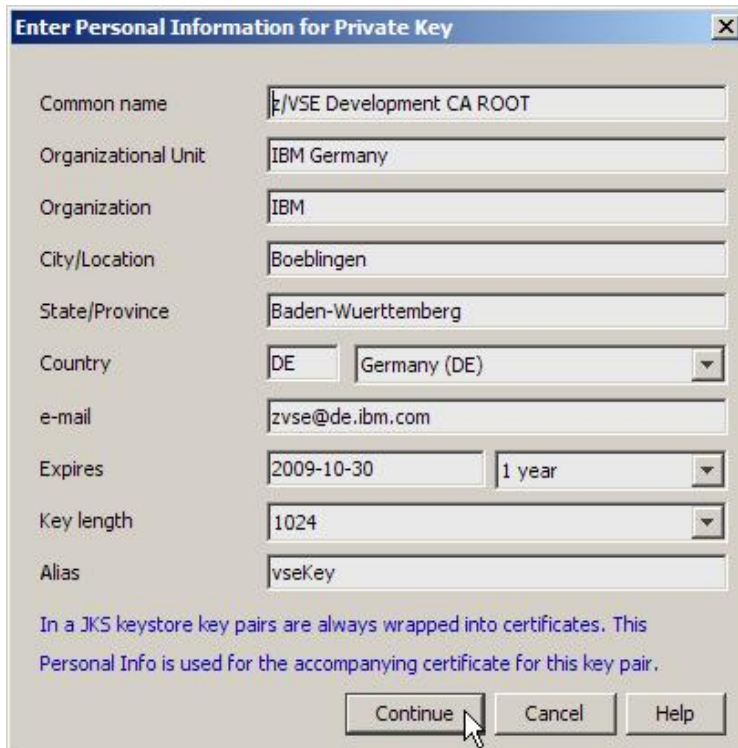
On the next dialog box, select JKS and enter a password for the Java keystore.



The dialog box titled "Local Keyring File - Properties" contains the following fields and controls:

- File Name: Keyring.JKS
- File Type: JKS PFX (with a "Browse..." button)
- Keyring File Password: *****
- Retype the password: *****
- JKS-specific options (Optional):
 - Private Key Password: [Empty field]
 - Retype the Priv. Key Password: [Empty field]
- Help text: "JKS keystores allow to optionally encrypt private keys with a different password than the file password. If not specified, the file password is used."
- Buttons: OK, Cancel, Help

Press **OK**.



The dialog box titled "Enter Personal Information for Private Key" contains the following fields and controls:

- Common name: /VSE Development CA ROOT
- Organizational Unit: IBM Germany
- Organization: IBM
- City/Location: Boeblingen
- State/Province: Baden-Wuerttemberg
- Country: DE (Germany (DE) dropdown)
- e-mail: zvse@de.ibm.com
- Expires: 2009-10-30 (1 year dropdown)
- Key length: 1024 (dropdown)
- Alias: vseKey
- Help text: "In a JKS keystore key pairs are always wrapped into certificates. This Personal Info is used for the accompanying certificate for this key pair."
- Buttons: Continue, Cancel, Help

As in a Java keystore, keys are always wrapped into certificates, you have to enter some personal information and press **OK**.



The Java keystore (Keyring.jks) can now be used directly by the z/OS Java Client.

5.2.4 Encrypt locally

Following command string encrypts a local file with the z/OS Java Client. Note that the alias name in Keyman must match with the `keyStoreCertificateAlias` parameter of the Java Client.

```
java -Djava.encryption.facility.debuglevel=1
com.ibm.encryptionfacility.EncryptionFacility
-mode encrypt
-underlyingKey AES16
-keyStoreName keyring.jks
-keyStoreType JKS
-keyStoreCertificateAlias vseKey
-password mypasswd
-inputFile mypic.jpg
-outputFile mypic.enc
```

Now upload the encrypted file to VSE.

```
ftp> put mypic.enc encdata
200 Command okay
150>About to open data connection
File:VSE.EF.ENCDATA
Type:Binary Recfm:FB Lrecl: 80 Blksize: 80
CC=ON UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON NAT=NO CONT=OFF
MODE=Stream STRU=File
150 File status okay; about to open data connection
226-Bytes received: 11,584
Records received: 145
Transfer Seconds: .02 ( 566K per second)
File I/O Seconds: .01 ( 1131K per second)
226 Closing data connection
ftp: 11584 bytes sent in 0.00Seconds 11584000.00Kbytes/sec.
```

We can now decrypt the file on VSE.

5.2.5 Do the decryption on VSE

In the JCL you have to specify the member name of the PRVK containing the private key (EFDECR) that was uploaded to VSE in section 5.2.2 on page 16.

```

* $$ JOB JNM=DECRSA,CLASS=4,DISP=D
// JOB DECRSA DECRYPT USING PRVK
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.PROD,PRD2.DBASE)
// EXEC IJBEFVSE
DECRYPT
RSA=CRYPTO.KEYRING(EFDECR)
CLRFIL=DD:PRD2.CONFIG(MYPIC.JPG)
ENCFIL=DD:ENCDATA
/*
/&
* $$ EOJ

```

6 Using record-based files

In the previous sections we used the z/OS Java client for encrypting a JPG picture, which has no record structure. However, a more realistic scenario is encrypting record-based files. We will see that line break characters need special attention when exchanging encrypted files between different platforms.

Before going to the examples below, a few words about the various file types we are dealing with.

1. **Stream-based data.** This is just binary data without any internal structure. Its representation on different platforms is exactly the same. However, there might be some difference in the underlying file systems: on workstation platforms we usually have flat files without any record structure. On VSE we for example have VSAM, where a binary stream may be split into multiple records without adding any meaning to it.

Encrypted files are always stream-based data!

2. **Line-based data.** The simplest example for line-based data is a plain text file with line break characters. On ASCII platforms there is a CRLF sequence (hex 0D0A) after each line, on EBCDIC platforms lines are split by a new-line character (hex 15). The important thing is that all lines are explicitly separated by line breaks.
3. **Record-based data.** Examples for record-based data are files that contain logical records, but don't have line break characters. One example are VSAM files, where each logical data record is stored in one physical record of the VSAM file. Another example are CMS files in z/VM where the length of each data record is given by the record length of the CMS file.

6.1 *Encrypting on z/OS*

When encrypting record- or line-based files on z/OS using the Java client, you must use the options `-recordFormat` and `-recordSize`. Refer to the "Encryption Facility for z/OS User's Guide, SA23-1349" for details on Java client parameters.

Following command string encrypts a data file with fixed 600 byte records.

```

java -Djava.encryption.facility.debuglevel=0
com.ibm.encryptionfacility.EncryptionFacility
-mode encrypt
-password mypasswd
-underlyingKey PBEWithSHA1AndAES16
-iterations 1000
-recordFormat FIXED
-recordSize 600
-inputFile testfile.txt
-outputFile testfile.crypt

```

If the z/OS file contains line breaks, you have to specify

```
-recordSize 601
```

Otherwise you might get the problem described in section “Misaligned records after decryption on VSE” on page 22.

6.2 *Decrypting on VSE*

When decrypting this file on VSE, we have two possible cases:

1. The target dataset on VSE (e.g. a VSAM file) is defined with LRECL=600. In this case the decryption on VSE will issue following warning message for each decrypted record:

```

WARNING: DATA EXCEEDS MAX RECORD LENGTH.
        RECORD IS TRUNCATED TO 600 BYTES.

```

Encryption Facility for z/VSE returns with rc=2 in this case.

2. The target dataset on VSE is defined with LRECL > 600. In this case the decrypted file will contain a hex 15 (the EBCDIC line break) at the end of each record.

6.3 *Encrypting on VSE*

Encrypting on VSE and decrypting on z/OS causes the situation that line breaks are missing on z/OS when the clear file on VSE didn't contain line breaks. The following example encrypts a file with 600 byte records without line breaks.

```

// EXEC IJBEFVSE
ENCRYPT
CLRTDES
PASSWORD=MYPASSWD
ICOUNT=1000
RECFM=F
LRECL=600
CLRFIL=DD:CLRDATA
ENCFIL=DD:ENCDATA
/*
/&

```

The options RECFM and LRECL are described in the z/VSE Administration Guide.

6.4 *Decrypting on z/OS*

When decrypting a record-based file using the Java client, it's not necessary to specify *-recordFormat* and *-recordSize*. The Java client, as well as Encryption Facility, maintains this information in the encrypted file. The following command line string would decrypt the above encrypted file with its original record structure.

```
java -Djava.encryption.facility.debuglevel=0
com.ibm.encryptionfacility.EncryptionFacility
-mode decrypt
-password MYPASSWD
-inputFile encrypted.file
-outputFile decrypted.file
```

But, as the dataset on VSE didn't contain line breaks, this will also produce a clear file on z/OS without line breaks. So you have to manually add line breaks for example by using an editor.

6.5 *EBCDIC to ASCII considerations*

When using the Java client on z/OS for encrypting an EBCDIC encoded data file, decryption on VSE will restore the clear file contents in EBCDIC-readable form. When decryption takes place on an ASCII platform, like a Windows PC, decrypted data is still encoded in EBCDIC. There is currently no option to specify some character set to be used when decrypting textual data.

7 Known problems

This chapter shows known problems from our test setup, but also from real customer installations.

7.1 *Misaligned records after decryption on VSE*

Symptom:

After decrypting a record-based file on VSE, the records are not aligned correctly. While the first record looks ok, the second record starts with a hex 15 byte. The third record has two additional bytes, where the second byte is hex 15, and so on.

```
Record 1 ...
15 Record 2 ...
xx 15 Record 3 ...
xx xx 15 Record 4 ...
```

Solution:

The EBCDIC new-line character is hex 15. The file was probably encrypted on another platform that supports line breaks, for example the z/OS Unix System Services shell. So the file was a plain Unix text file with, let's say, records of 600 bytes, the new-line character not counted.

Most likely the file was encrypted with options `-recordFormat F` and `-recordSize 600`, but the records in fact have 601 bytes, which causes this misalignment.

You can solve the problem by

- Either using options `-recordFormat F` and `-recordSize 601`, (in this case you will get records ending with a hex 15 after decrypting on VSE if the LRECL of your VSE dataset is big enough), or by
- Removing the new-line characters from the file before encrypting it.

7.2 Using CA DynamT

Symptom:

When using CA DynamT to open a clear tape or vtape, Encryption Facility loops endlessly. The TAPESRVR job starts, but there is no apparent I/O against the vtape file, once the file open sequence has completed. JCL similar to the following is used:

```
// EXEC TDYNASN
OPEN CLRDATA ,SYS005 ,INPUT
/*
```

Solution:

Adding the 'P' option to the TLBL for CLRDATA should solve the problem.

```
// TLBL CLRDATA , 'RAID23 ,U ,P'
```

Dynam/T opens the VTAPE successfully. However Dynam/T performs dynamic LUB allocation and does not assign the VTAPE to SYS005. In this example, Encryption Facility is expecting CLRDATA to be assigned to SYS005.

8 More information

More information can be found on these web pages.

VSE Homepage

<http://www.ibm.com/servers/eserver/zseries/zvse/>

Keyman/VSE tool and VSE Connector Client

<http://www.ibm.com/servers/eserver/zseries/zvse/downloads/>

Encryption Facility for z/OS

http://www.ibm.com/servers/eserver/zseries/zos/encryption_facility/

Encryption Facility for z/OS publications

<http://publib.boulder.ibm.com/infocenter/zos/v1r9/index.jsp?topic=/com.ibm.zos.r9.e0zc100/e0z2c18038.htm>

IBM Encryption Facility for z/OS Java Client

<http://www.ibm.com/servers/eserver/zseries/zos/downloads/#efclient>

Redbook: Encryption Facility for z/OS Version 1.10, SG24-7318

<http://www.redbooks.ibm.com/abstracts/sg247318.html?Open>

Keytool parameters

<http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html>

KeePass Password Safe – a free Open Source Password Manager for many operating systems

<http://keepass.sourceforge.net/>

VSE Virtual Tape Server

<http://www.ibm.com/servers/eserver/zseries/zvse/downloads/>

RSA Security - PKCS #5: Password-Based Cryptography Standard

www.rsasecurity.com/rsalabs/pkcs/pkcs-5/

PKCS #5: Password-Based Cryptography Specification, Version 2.0

<http://tools.ietf.org/html/2898>