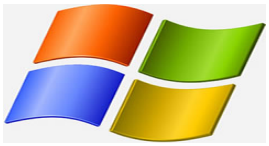
CryptoXpress™ LT

An affordable, commercial grade, "strong" encryption solution intended for users who need a ready-to-run solution.



Installation and User Guide

Software Version	9.0
Document	CryptoXpressLT.pdf
Release Date	07/14/2007

CFXWorks, Inc.
5365 Chelsen Wood Drive, Duluth, Georgia 30097

Email: sales@cfxworks.com http://www.CFXWorks-Enterprise.com

Printed in the United States of America.

Minimum System Requirements

The **CryptoXpress**™ family of offerings have been tested on IBM server platforms as well as Windows, Linux (Red Hat and SUSE), HPUX 10.0 and Solaris 8.0 platforms. The Windows version should function properly on all Win32 platforms. The Linux version should function on all versions of Linux. The HPUX version should function on HPUX 10.0 and all subsequent versions. The Solaris version should function on Solaris 8 and all subsequent versions. On all the above systems Java Version 1.4.0 or a more current version of Java is required. On IBM's iSeries running OS/400, Java Version 1.4.2, or a more current version of Java, is required.

Software License

Demonstration copies of **CryptoXpress LT** are available. Demo copies contain expiration dates. Purchased copies of **CryptoXpress LT** have no expiration date.

Technical Limitations

The AES and 3DES algorithms have not been modified in any way from those published by the NIST.

There is no limitation imposed by CFXWoek's software on the size of files or message strings that can be processed. **CryptoXpress LT** has been tested extensively on message strings from 0-65536 characters in length and for file sizes up to 2 Mbytes in size. Messages or files exceeding these values may function properly however, **CryptoXpress LT** has not been tested beyond these limitations.

Export Limitations

CryptoXpress LT contains encryption technology that is subject to the U.S. Export Administration Regulations and other U.S. laws and may not be exported or re-exported to certain countries (currently Afghanistan (Taliban-controlled areas), Cuba, Iran, Iraq, Libya, North Korea, Serbia (except Kosovo), Sudan and Syria) or to persons or entities prohibited from receiving U.S. exports (including Denied Parties, entities on the Bureau of Export Administration Entity List, and Specially Designated Nationals). For more information on the U.S. Export Administration Regulations http://www.bxa.doc.gov/Encryption/regs.htm, 15 C.F.R. Parts 730-774, and the Bureau of Export Administration U. S. Department of Commerce. Please see the home page www.bxa.doc.gov

Support

Support is provided for this offering via email addressed to support@cfxworks.com.

Warranty

Please read the license file "license_CryptoXpressLT.pdf" in the distribution zip file.

Trademarks and Copyrights

CryptoXpress™ is a registered trademark of CFXWorks, Inc.

IBM® is the trademark of International Business Machines Corporation in the United States, other countries, or both.

Java® and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Intel®, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

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

1. INTRODUCTION

1.1. Thank You!

Thank you for purchasing **CryptoXpress™ LT** offering. Our intent is to provide an easy to use offering that supports "strong encryption" capabilities as defined by the National Institute of Standards and Technology (NIST). Please address questions and feedback via email to support@cfxworks.com.

CFXWorks intends to provide a family of cryptography offerings that includes:

- **CryptoXpress™ SDK –** This is the core toolkit from which we build all of our crypto offerings. We make it available as a Software Development Toolkit (SDK). It is intended for use by skilled Java programmers. Note that prior to Release 9.0, this product was called CRYPTOeServer.
- **CryptoXpress™ LT – This is a collection of ready to run Java programs that support "strong encryption" and message digests. This product is currently available.**
- **CryptoXpress™ SOA–** This is a SOA implementation that supports "strong encryption" and message digests. It is intended to [provide cryptography services to any program or user using HTTP/HTTPS protocols as the interface. One of the major advantages of using an SOA service to deliver cryptography services is that it provides a solution that gurantees compatibility and consistency across any program residing on any system that has HTTP/HTTPS access to the service. Note that prior to Release 9.0, the SOA capability was included in CRYPTOeServer. The new version of this product will be available 3rd quarter 2007.
- **CryptoXpress™ ColdFusion–** This is ColdFusion tag (a plugin) that provides a collection of ready to run cryptography functions that can be used by ColdFusion programmers. This product is planned for availability 3th quarter 2007.
- **CryptoXpress™ .Net–** This is a collection of ready to run C/C++ cryptography programs that target .Net users. It will also provide a (C/C++) interface to **CryptoXpress™ SOA.** This product is planned for availability 4th quarter 2007.
- **CryptoXpress™ DMZ –** This solution address many of the security concerns related to securing data that resides within a

DMZ and the secure transport of data to and from a DMZ from within the Intranet. Custom versions of this product have been delivered to several government agencies over the past several years. This version of the product is planned for availability 1st quarter 2008.

- **CryptoXpress™ 400–** This is a collection of ready to run C/C++ cryptography programs that target IBM's iSeries (AS/400). It will also provide a (C/C++) interface to **CryptoXpress™ SOA.** This product is a variation of our current DataQueueCrypto400 product. The new version will not require data queues. It is planned for availability 1st quarter 2008.

1.2. General Description

CryptoXpress LT provides several Java programs that can be used to encrypt/decrypt text or files. It also provides Java programs that can digest, sometimes referred to as signing, either text or files. **CryproXpress LT** produces compatible results across many platforms including Windows, Linux (Red Hat), HPUX, Solaris, and IBM's iSeries running OS/400. **CryptoXpress LT** should run successfully on any Java enabled platform using either Sun's or IBM's JVM. Java Version 1.4, or a more current version of Java, is required.

1.3. Data Confidentiality (encryption)

CryptoXpress LT can be used to encrypt and decrypt text or files. It supports both the AES and 3DES (TripleDES) encryption algorithms. AES is a block cipher (symmetric key) encryption algorithm that supports 128-bit, 192-bit and 256-bit key sizes. **CryptoXpress LT** supports 128-bit and 256-bit key sizes

On May 19, 2005, NIST announced the withdrawal of the (single) Data Encryption Standard (DES) as specified in FIPS 46-3. DES no longer provides the security that is needed to protect Federal government information. Federal government organizations are now encouraged to use FIPS 197, Advanced Encryption Standard (AES), which specifies a faster and stronger algorithm. For some applications, Federal government departments and agencies may use the Triple Data Encryption Algorithm (3DES or TripleDES) as specified in NIST Special Publication 800-67. 3DES is also supported by **CryptoXpress LT**. Although thought to be considerably less secure than AES 128-bit encryption, 3DES is still commonly used in some industries.

1.4. Data Integrity (message digests)

CryptoXpress LT can be used to calculate a message digest for data or files. The act of calculating a message digest is sometimes referred to as "digesting" the information. The result of a message digest is sometimes referred to as a digital signature.

A message digest (also sometimes referred to as a one-way hash function) is a fixed length, computationally unique identifier corresponding to a set of data. The result of the algorithm is that each file or data string digested will map to a particular block of information called a message digest. The digest is not random; digesting the same unit of data with the same algorithm will always produce the same message digest.

Most users prefer to use the MD5 message digest algorithm. MD5 belongs to a family of one-way hash functions called message digest algorithms. The MD5 system is defined in RFC 1321. MD5 takes a message of arbitrary length and produces as output a 128-bit message digest. It is conjectured that it is computationally infeasible to produce two different messages having the same message digest, or to produce any message having a given message digest. RFC 1321 also defines a certification suite to validate correct implementation of the algorithm. **CryptoXpress LT** is validated against this suite.

1.5. Common Uses for Encryption

The following list defines some of the current regulations and legislation either requiring or suggesting the use of "strong encryption".

- Cardholder Information Security Program (CISP) *(very important for merchants conducting e-Commerce transactions over the web)*
- Payment Card Industry Data Security Standard (PCI) *(very important for merchants conducting e-Commerce transactions over the web)*
- The Sarbanes Oxley Act (SOX)
- The Gramm-Leach-Bliley Act, the Safeguards Rule (GLBS)
- Health Insurance Portability and Accountability Act (HIPPA)
- California Assembly Bill 1950 (AB 1950)
- Title 21 of the Federal Regulations Part 11 (21 CFR part 11)
- California Information Practice Act or Senate Bill 1386
- North American Electric Reliability Council (NERC)
- Federal Information Security Management Act (FISMA)
- USA PATRIOT Act

- Cardholder Information Security Program (CISP)
- Payment Card Industry Data Security Standard (PCI)
- Federal Information Processing Standards (FIPS)
- National Association of Securities Dealers Rule 2711
- SEC 17a-4

Encryption is commonly used where it is necessary to transmit or store sensitive information. The following includes examples of data that is commonly encrypted.

Credit Card Information	***Personal Information***
Card number	Address information
Name on credit card	Phone numbers
CVV2 data on card	Service numbers
Card expiration date	Social security numbers

Employee Data	***Medical information***
Contact information	Age
Salary data	Medical history
Performance data	Medication

1.6. Common Uses for Message Digests

Message digests have many uses. In particular, they are used to authenticate data. For example, to create a digest for authentication, data can be digested and the digest saved. Later, to validate that the data has not been altered, the data is digested again and the result is compared against the original digest. If they differ, the data has been altered. This is very different from encryption because the actual data is not modified when it is digested. Encryption is intended to protect the confidentiality of data. A message digest is used to assure data integrity.

2. ENCRYPTION/DECRYPTION

2.1. Encryption Algorithms Supported

For ease of use, **CryptoXpress LT** simplified the selection process by reducing thousands of possible combinations of encryption algorithms, key sizes, modes of operation, and paddings to the following “preferred” combination:

1. 128-bit encryption AES128/PKCS5Padding/CBC
2. 128-bit encryption AES256/PKCS5Padding/CBC
3. TripleDES encryption 3DES/PKCS5Padding/CBC

These combination are ones for which the NIST has published test vectors. CFXWorks has tested the results of our implementation using these test vectors.

2.2. Why use AES Encryption?

There are several good reasons to use AES encryption versus other encryption algorithms:

- The Federal Government has defined a new standard for encrypting electronic documents and messages, a code so secure that federal officials predict that its encoded material will remain secure for 20 to 30 years. This code, the Advanced Encryption Standard (AES), received formal approval from Commerce Secretary Donald Evans on December 4, 2001. AES (formally called Rijndael) replaces the Data Encryption Standard (DES). DES, originally adopted in 1977, can now be deciphered with modern computers and decryption methodologies. **The Federal Government now requires all agencies within, suppliers to, and contractors and sub-contractors to the federal government use the AES encryption algorithm.**

- The performance characteristics and form factor (code size) of AES is superior to most other 128-bit algorithms.

- The security level of AES is thought to be superior to other commercially available encryption algorithms.

- Most of the more secure encryption algorithms have historically been patented technologies. Therefore, they tend to be very expensive. License fees in excess of over $200,000 per system are not uncommon. Negotiating licenses for products from some vendors have been historically near impossible for small businesses. AES is not patented.

2.3. How secure is AES?

To put this issue in perspective, here are some statistics presented by the National Institute of Standards and Technology (NIST) relative to the possibility that someone could crack a 128-bit Rijndael encryption key.

http://www.nist.gov/public_affairs/releases/aesq&a.htm

"In the late 1990s, specialized "DES Cracker" machines were built that could recover a DES key after a few hours. In other words, by trying possible key values, the hardware could determine which key was used to encrypt a message. Assuming that one could build a machine that could recover a DES key in a second (i.e., try 255 keys per second), then it would take that machine approximately 149 thousand billion (149 trillion) years to crack a 128-bit AES key. To put that into perspective, the universe is believed to be less than 20 billion years old."

2.4. What if I forget my encryption key?

☑ **If you forget you encryption key, you should assume that your data is irretrievably lost.**

AES is a **very serious** encryption algorithm. In our lifetime, hackers are not likely to be able to compromise this algorithm. Also, there are no known back doors to this algorithm. If you forget your encryption key, there is absolutely no way that our organization, or any other organization known to exist, can retrieve it.

Although 3DES is somewhat less secure than AES 128-bit encryption, it is still secure enough that if you forget you encryption key, you should also assume that your data is irretrievably lost.

2.5. Do all AES solutions produce compatible results?

The answer is emphatically no! The specifications for nearly all encryption algorithms allow the implementer to choose from many options, such as key size, block size, mode of operation, packing and key manipulation techniques. The result is that there are several thousand different combinations that can be deployed, each yielding different results. To produce compatible results you have to know precisely what options and techniques the vendor implemented and duplicate those options and techniques. That is why it is wise to select a vendor who specifically supports all ot the platforms you need a solution for.

2.6. When should I use 3DES?

3DES is considerably slower and significantly less secure than even 128-bit AES encryption. 3DES is thought to be equivalent to approximately 112-bit encryption. However, some institutions are heavily invested in equipment that uses 3DES (for example, the banking and credit card industry are heavy users of 3DES.)

☑ For compatibility purposes, **CryptoXpress LT** supports 3DES. But if the decision is yours, always use AES.

2.7. What about ASCII versus EBCDIC?

Encryption algorithms treat data as binary. Therefore it makes no difference to the algorithm what encoding scheme was used to capture or display the data. Data captured and encrypted on an ASCII machine can be decrypted on an EBCDIC machine and vice versa. The binary values should compare exactly. However, as you might expect, binary data is likely to display different on an ASCII machine then it would on an EBCDIC machine.

2.8. Modes of operation

When encrypting data, there are two popular modes of modes of operation: ECB (Electronic Block Mode) and CBC (Cipher Block Chaining) mode. With ECB mode, the cipher takes a single block of plaintext and produces a single block of ciphertext. Data streams are broken into blocks that are individually processed. Each block is 16 bytes long.

☑ **CryptoXpress LT supports the CBC mode.** The CBC mode is considered more secure than the ECB mode for encrypting messages over one block long (16 bytes). In CBC mode, the plaintext is XORed with the previous ciphertext block before it is encrypted. After a plaintext block is encrypted, the resulting ciphertext is stored in a feedback register. Before the next plaintext block is encrypted, it is XORed with the feedback register to become the next input to the encrypting routine. The resulting ciphertext is again stored in the feedback register, to be XORed with the next plaintext block, and so on until the end of the message. The encryption of each block depends on all previous blocks. Each block is 16 bytes long.

2.9. Padding

☑ The length of data to be decrypted may not be an even multiple of the block size. Therefore the cipher pads short blocks. Padding is added when the data is encrypted. Padding is removed when the data is decrypted. There are several padding techniques. **CryptoXpress LT uses a very common technique called PKS5Padding.**

2.10. Initialization Vectors (Salt)

When using CBC, you must supply both a key and an initialization vector (sometimes called a salt). The initialization vector (IV) is used to seed the feedback register prior to encrypting the first block of data. The IV has no meaning; it is just there to make each message unique. The IV value must be supplied to both the encryption and decryption routine. The IV need not be secret, however, you must remember the IV value, just as you must remember you encryption key.

☑ The IV value for AES is 16 bytes long. The IV value for 3DES is 8 bytes long. If you supply a value shorter than the required length, **CryptoXpress LT fills out the IV to the required length with 0x00. If you supply a value over the required length, CryptoXpress LT truncates the right most excess bytes.**

2.11. Key Size

The AES algorithm requires a 16 byte key for 128-bit encryption and a 32-byte key for 256-bit encryption. 3DES requires a 24-byte key.

☑ If you supply a value shorter than the required key size, **CryptoXpress LT fills out the key to the required length with 0x00. If you supply a value over the required length, CryptoXpress LT truncates the right most excess bytes.**

2.12. Length of encrypted data

AES is a block cipher. Therefore encrypted data will generally be longer that unencrypted data. If encrypted data is to be stored in a database, the column within the database must support the length of the encrypted data.

Encryption algorithms generally round the length of data up to a block boundary. For AES this means that encrypted data will be rounded up to a 16 byte boundary. Data that is an even multiple of 16 bytes long will have an additional 16 bytes added to the data.

For example, data 12 bytes long when encrypted using AES will be 16 bytes long. Data 16 bytes long when encrypted will be 32 bytes long. Data 20 bytes long when encrypted will be 32 bytes long.

3DES uses a block size of 8. Therefore it rounds data up to an 8 byte boundary. Therefore, data 6 bytes long when encrypted using 3DES will be 8 bytes long. Data 12 bytes long when encrypted will be 16 bytes long. Data 16 bytes long when encrypted will be 24 bytes long.

2.13. Best Practices

The probability that anyone would be capable of directly compromising the integrity of the AES algorithm is thought to be next to zero. Please refer to the comments in Section 2.3 of this document. However, the integrity of any encryption algorithm relies on the user to protect the confidentially of the encryption key and to select keys that are not easily guessed. For example, if user “John Doe” selects as his encryption key “john”, it isn’t going to take long to guess the password.

☑ As a rule of thumb, select a key that is the maximum length allowed. For example, for 128-bit encryption, select a key that is 16 characters long. The safest key would be a key containing a mix of upper case alpha, lower case alpha, numeric and special characters. This combination makes the key very difficult to guess.

3. MESSAGE DIGESTS

3.1. Message Digests

Encryption is intended to protect the confidentiality of data. However, how do you determine if a black hat (bad guy) has changed the contents of an encrypted data string or data file? Changing data content relates to data integrity, not data confidentiality. The solution to this issue is what cryptologist call a message digest .A message digest is sometimes called a digital signature.

CryptoXpress LT supports the following message digest algorithms:

- MD5 – MD5 digests either a data string or a file and creates a 16 byte message digest. **CryptoXpress LT** returns the digest in binary and as a hex string. The binary value is 16 bytes long. The hex string is 32 bytes long.

- SHA1 – SHA1 digests a string and creates a 20 byte message digest. **CryptoXpress LT** returns the digest in binary and as a hex string. The binary value is 20 bytes long. The hex string is 40 bytes long.

3.2. Using Message Digests

A common use of a message digest is to construct a digital envelope. A digital envelope is constructed as follows:

A message digest is calculated for a string of data and then concatenated to the string. Then, the total string is encrypted. This forms a digital envelope. When the data is decrypted, the message digest is recalculated and compared to the original value. If even a single bit within the encrypted string or file has been modified, the comparison fails. There are many variations to this theme but cryptologists commonly use this technique to transport data since it is considered the most secure way known to transport information form one point to another.

3.3. How Secure Are Message Digests?

It is said that the difficulty of coming up with two messages having the same MD5 message digest is in the order of 2^64. The difficulty in defining a message with a specific message digest is in the order of 2^128.

SHA1 is believed to be significantly more secure than MD5, but it is much slower.

3.4. Which Message Digest Algorithm Should I Use?

MD5 is probably the most popular message digest algorithm in use today because it offers a reasonable balance between performance and security. If you are doing work for the DOD, they will probably require you to use SHA1.

You can learn more about MD5 at http://www.nic.mil/ftp/rfc/rfc1321.txt. You can learn more about SHA1 at www.itl.nist.gov/fipspubs/fip180-1.htm.

Note that to use the SHA1 digest routines you must have the strong encryption policy files installed on your system. See Section 4.8 paragraph number 3 for details.

4. PRODUCT INSTALLATION

4.1. Minimum System Requirements

CryptoXpress LT has been tested on all IBM eServer platforms as well as Windows 2000, Linux (Red Hat), HPUX 10.0 and Solaris 8.0 platforms. The Windows version should function properly on all Win32 platforms. The Linux version should function on all versions of Linux. The HPUX version should function on HPUX 10.0 and all subsequent versions. The Solaris version should function on Solaris 8 and all subsequent versions.

On all the above systems Java Version 1.4.0 or a more current version of Java is required. On IBM's iSeries running OS/400 Java 1.4, or a more current version of Java, is required.

4.2. CryptoXpress LT Components

The following jar files are included in the distribution zip file ("CryptoXpress.zip"):

File	Description
CryptoXpressLT.jar	Required jar file that contains all the CryptoXpress java code.
log4j-1.2.14.jar	Required jar file that contains the log4j code.
log4j.properties	Configuration file used for logging.
local_policy.jar	IBM version of the strong encryption version of this jar file (1)
US_export_policy.jar	IBM version of the strong encryption version of this jar file (1)

4.3. CryptoXpress LT Sample Scripts

The following script files are included in the distribution zip file ("CryptoXpress.zip"):

Script File	Description
t1.bat or xt1 – sample program (2)	This script reads the command line arguments passed to CFXCMD and pipes them to CFX103TF. CFX103TF encrypts the data passed and writes the encrypted data to a file. CFXF2D is used to display the encrypted file in HEX.
t2.bat or xt2 – sample program (2)	This script reads an input file using CFX103FF, encrypts it, and writes the output to a file. CFXF2D is used to display the encrypted file in HEX.
t3.bat or xt3 – sample program (2)	This script reads the command line arguments passed to CFXCMD and pipes them to CFXT2F which creates an output file. CFXF2D is use to display the output file in HEX. CFX103FF is then used to encrypt the file and write the encrypted data to another output file. CFXF2D is used to display the encrypted file. CFX103FF is then used to decrypt the file. CFXF2D is used to display the unencrypted file in HEX.
t4.bat or xt4 – sample program (2)	This script uses CFX103FF to encrypt a file and write the encrypted data to another output file. CFXF2D is used to display the encrypted file. CFX103FF is then used to decrypt the file. CFXF2D is used to display the unencrypted file in HEX.
t5.bat or xt5 – sample program (2)	This script reads the command line arguments passed to CFXCMD and pipes them to CFXT2F which creates an output file. It then uses CFXConvert to convert this file from binary to ASCII. It then uses CFXF2D to display the output file in HEX. Next, it attempts to convert the file from ASCII to EBCDIC. Finally it uses CFXF2D to display the final file in HEX.
t6.bat or xt6 – sample program (2)	This script reads the command line arguments passed to CFXCMD and pipes them to CFX104TF. CFX104TF encrypts the data passed and writes the encrypted data to a file. CFXF2D is used to display the encrypted file in HEX.
t7.bat or xt7 – sample program (2)	This script reads the command line arguments passed to CFXCMD and pipes them to CFX112TF. CFX112TF encrypts the data passed and writes the encrypted data to a file. CFXF2D is used to display the encrypted file in HEX.
t8.bat or xt8 – sample program (2)	This script reads the command line arguments passed to CFXCMD and pipes them to CFX401T. CFX401T digests the data using MD5 and writes the digest in HEX to the standard output device.
t9.bat or xt9 – sample program (2)	This script reads the command line arguments passed to CFXCMD and pipes them to CFX402T. CFX402T digests the data using SHA1 and writes the digest in HEX to

Script File	Description
	the standard output device.

(1) These files are required when using 128-bit or 256-bit encryption. They are also required to use SHA1. If you are running Sun's Java Virtual Machine you will require Sun's version of these files. If you are running IBM's Java Virtual Machine you will require IBM's version of these files.

If you are running Sun's Java 1.4 implementation, you generally need Sun's strong encryption policy files for 1.4. A copy of Sun's policy files can be downloaded from the Sun web site at http://java.sun.com/products/jce/index-14.html

If you are running Sun's Java 1.5 iimplementation, you generally need Sun's strong encryption For example, you can download Java 5.0 and the strong encryption policy files at http://java.sun.com/javase/downloads/index_jdk5.jsp. To download the runtime version of Java download "Java Runtime Environment (JRE) 5.0 Update 12. files from 1.5. To download the strong encryption files download under "Other Downloads"… Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 5.0.

If you are running Sun's Java 1.6 implementation, you generally need Sun's strong encryption policy files for 1.6. A copy of Sun's policy files can be downloaded from the Sun web site at http://java.sun.com/javase/downloads/index.jsp.

If you are running IBM's Java Virtual Machine you will need IBM's strong encryption policy files. IBM's site for Java downloads is at http://www.ibm.com/developerworks/java/jdk/.

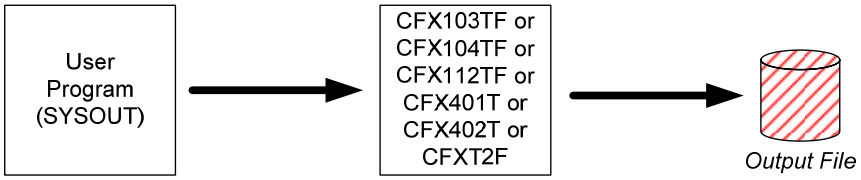
(2) The .bat version is for Windows. The x version is for Linux, UNIX or the OS/400.

4.4. CryptoXpress LT Programs

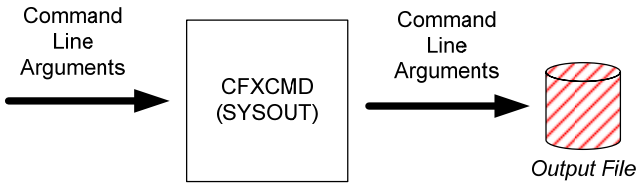
The **CryptoXpress LT** programs documented in this section of the manual are of the following form.

1. The programs read input from a file and write output to a file:

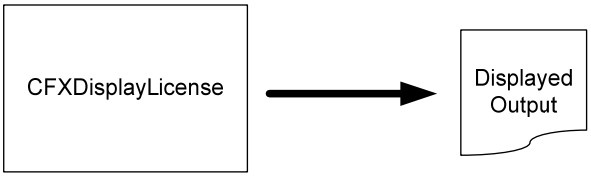
2. The programs read input from the standard input device (sysin) and write output to a file. This capability is especially useful when one wants to pipe output from one programs standard output and read it as input by the program performing the crypto task:



3. The program reads a command line argument and writes them to standard output:



4. The program simply displays output to the standard output device:



The following Java programs are included in CryptoXpress:

Java Application Examples	Description
CFXCMD	This program can be used to read command line arguments and pipe them to programs that read data from the standard input device (sysin). CFXCMD reads command line arguments and writes them to the standard output device (sysout). Note that this program outputs data exactly as it is interpreted by the operating system's command line parser. Depending on the operating system, it is impossible to pass some characters within command line

Java Application Examples	Description
	arguments. Some characters are likely to be ignored and some are interpreted differently than you might expect. Therefore test your input carefully if you intend to use this program.
CFX103FF	A Java program that can be used to encrypt or decrypt a file using 128-bit AES/PKCS5Padding/CBC and writes the output to a file.
CFX103TF (1)	A Java program reads data from the standard input device (sysin), encrypts it using 128-bit AES/PKCS5Padding/CBC and writes the output to an encrypted file.
CFX104FF	A Java program that can be used to encrypt or decrypt a file using 256-bit AES/PKCS5Padding/CBC and writes the output to a file.
CFX104TF (1)	A Java program reads data from the standard input device (sysin), encrypts it using 256-bit AES/PKCS5Padding/CBC and writes the output to an encrypted file.
CFX112FF	A Java program that can be used to encrypt or decrypt a file using 3DES and writes the output to a file.
CFX112TF (1)	A Java program reads data from the standard input device (sysin), encrypts it using 3DES and writes the output to an encrypted file.
CFX401F	A Java program that reads an input file, digests it using MD5, and writes the output to the standard output device (sysout).
CFX401T (1)	A Java program reads data from the standard input device (sysin), digests it using MD5, and writes the output to the standard output device (sysout).
CFX402F	A Java program that reads an input file, digests it using SHA1, and writes the output to the standard output device (sysout).
CFX402T (1)	A Java program reads data from the standard input device (sysin), digests it using SHA1 and writes the output to the standard output device (sysout).
CFXF2D	A Java program that reads a file and displays if in HEX to the standard output device (sysout).
CFXT2F (1)	A Java program reads data from the standard input device (sysin) and writes the output to the standard output device (sysout).
CFXDisplayLicense	Displays the CryptoXpress license information to the standard output device (sysout).
CFXConvert	Converts a file from ASCII to EBCDIC or from EBCDIC to ASCII.

(1) Programs that read data from the standard input device (sysin) need to know when end of file (end of input) is reached. How this is done is platform dependent. On some systems Ctrl-D when entered from the keyboard is used. On Windows 2000 the “Enter” is used to trigger end of file. A Line Feed (Decimal 10, 0x0a) or a Carriage Return (Decimal 13, 0x0d) in the input stream also causes end of file. Other than 0x0a and

0x0d, all other binary characters from 0x00 – 0xff are read by the standard input device without change or interpretation.

4.5. Documentation

The following documentation ships with **CryptoXpress**:

File	Description
CryptoXpress.pdf	This document
License_CryptoXpress.pdf	License file.

4.6. ASCII and EBCDIC Conversions

Data captured and recorded on different platforms may be stored in different encoding formats. Depending on what you intend to do with the data you may have to translate it from one format to another. For example, if you capture data on a machine that records it in ASCII format, the data will not display correctly on a machine that records data internally in EBCDIC format. For display purposes you will have to translate it from ASCII to EBCDIC before you display it.

The CFXConvert sample program is provided to convert file encoded in one format to another. The following command will convert the file named input.txt from ASCII to EBCDIC:

```
CFXConvert /f126 /iinput.txt /ooutput.txt
```

The following command will convert the file named input.txt from EBCDIC to ASCII:

```
CFXConvert /f127 /iinput.txt /ooutput.txt
```

Note that these programs will not run on a system using Sun's Java Virtual Machine. They require IBM's Java Virtual machine.

4.7. Command Line Parameters

The input parameters supported for the sample Java programs are illustrated in the following table.

Parameter	Description	Sample
/a	Encrypt or decrypt data (1)	/ae encrypt /ad – decrypt
/i	The fully qualified name of the input file (1)	/iinput.txt

/o	The fully qualified name of the output file (1)	/ioutput.txt
/k	The key to use for encryption/decryption. (1)	/k123
/v	The initialization vector to use for encryption.decryption. (1)	/v456
/f	Data fomats or conversions.	/f126 convert ASCII to EBCDIC (3) /f127 convert EBCDIC to ASCII (3)
/b	Name of key file.	Not currently supported.
/p	Password for key file.	Not currently supported.
text	Input text. (2)	1234567890123456

(1) No defaults are assumed for any of these values.
(2) If a text value contains blanks, enclose the text in quotes as follows: "This is the text that I want to encrypt"
(3) These options are not available on the Sun Java Virtual machine. They are supported on the IBM Virtual machine.

The following parameters are supported for each sample program:

Parameter	/a	/i	/o	/k	/v	/f	Text
CFXCMD							
CFX103FF	X	X	X	X	X		
CFX103TF	X		X	X	X		X
CFX104FF	X	X	X	X	X		
CFX104TF	X		X	X	X		X
CFX112FF	X	X	X	X	X		
CFX112FF	X		X	X	X		X
CFX401F		X					
CFX401T							X
CFX402F		X					
CFX402T							X
CFXF2D		X					
CFXT2F			X				X
CFXDisplayLicense							
CFXConvert		X	X			X	

4.8. Error Codes

RC - description
0 - OK
-4 - Invalid padding
-5 - Zero length input
-6 - Invalid key
-7 - Illegal block exception
-8 - Bad padding exception
-9 - Invalid algorithm parameter exception (1) (2)
-10 - Missing filename

-14 - File not found
-18 - Invalid IV length
-19 - Invalid Base64 encoding
-90 - License error... invalid digital signature
-92 - License error... time expired
-93 - License error... error creating license file
-97 - Zero length input.
-98 - Invalid command line option
-99 - Encryption/Decryption error (trying to decrypt an clear input)

(1) The CFXConvert program will display this error message if run using the Sun's Java Virtual Machine because Sun does not support the EBCDIC conversion algorithms.
(2) If the encryption programs are run on a PC that does not have the correct strong encryption policy files installed (see Section 4.8), this error code will be displayed.

4.9. Windows, Linux & UNIX Installation

The process is outlined below. Note that **Java, UNIX and Linux are case sensitive when using class and jar file names**. Failure to honor this requirement is most common cause of installation failure.

1. The first step in the installation process is to install java on the system on which these **CryptoXpress LT** programs will execute. **CryptoXpress LT** requires either java 1.4.1 or a more recent version of Java. If your system is a Windows. Linux or UNIX system we suggest that you install either the run-time version of Java 1.5 or the SDK. If you are running OS/400 we suggest that you install java 1.4.2.

   You can verify that you have installed Java correctly by running the following command:

   Java -Version

   The system should respond with a message similar to the following:

   Java version "1.4.2"

2. **CryptoXpress LT** is distributed in a zip file ("CryptoXpress.zip") that contains several Java jar files and several sample Java programs and this document. Unzip these files into a directory of your choice.

Note that the sample scripts assume that they are being executed with the current working directory being the **CryptoXpress LT** installation directory. The scripts can be changed to point to any directory.

3. **You must copy the "strong encryption" jar files available from either Sun or IBM to the appropriate directory on the host system (see discussion in section 4.2)**. These files, (local_policy.jar and US_export_policy.jar) must be installed in the **%JAVA_HOME%/jre/lib/security** directory. They replace files of the same name that are distributed in the standard Java SDK. The standard distribution files that come with Java do not allow the use of strong encryption. **On an iSeries platform the installation directory for these policy files is generally /QIBM/Proddata/Java/jdk14/lib/security.**

   Note that the DQAim400 zip file contains only the IBM version of these policy files. The Sun versions of these files must be downloaded from Sun. See comment (1) in Section 4.2 of this manual.

   On a Windows system "JAVA_HOME" can be found by entering the command:

   Echo %JAVA_HOME%

   On a Solaris or Linux system "JAVA_HOME" can be found by entering the command:

   Echo $JAVA_HOME$

4.10. iSeries Special Installation Instructions

Some versions of the OS/400 system have backlevel or multiple levels of Java installed. This may make installation of **CryptoXpress LT** confusing. **CryptoXpress LT** requires Version 1.4.x, or a more current version of Java.

To run the Java programs you can use the RUNJVA command or execute the program from the Qshell. The scripts assume that the Java programs are being run under qshell. Qshell is accessed by running the following command:

qsh

