A Guide To TclDES

Munitions-grade Tcl Scripting!

Version 0.8

Table of Contents

Illustration Index

Index of Tables

Introduction

This guide provides instruction on the setup and use of TclDES and TclDESjr, which are pure-Tcl implementations of the National Institute of Standards and Technology (NIST) Data Encryption Standard (DES). This guide will not go into a detailed analysis of the operation of DES. Many documents are availble that provide that information. Rather, this guide describes the various operational modes of DES and how the procedures in TclDES and TclDESjr are invoked to achieve these modes. Issues regarding the import and export of TclDES will also be discussed.

What Is DES?

In 1972, the National Bureau of Standards (NBS), now known as NIST, issued a call for proposal for a government-wide data encryption standard to be used for secure communications and data protection of unclassified, sensitive information. IBM responded with a modified version of the "Lucifer" encryption algorithm known as the Data Encryption Algorithm (DEA). A slightly modified form of this algorithm was adopted in 1976 as DES, and was published as FIPS PUB 46 in 1977. The standard is currently in its third revision [1]. DES is still known as DEA in the American National Standards Institute (ANSI) standard publication ANSI X3.92-1981. A good historical background of DES is presented in [2]. A good source for general information about cryptography, including DES is presented in [3].

This guide will not provide a detailed presentation of the DES algorithm, as there are many descriptions and analyses of the algorithm available openly that satisfy that need, such as [4] and [5]. Here, it will be sufficient to treat the DES algorithm as a black-box function with specific inputs and outputs that have specific characteristics.

The DES algorithm is a block cipher, which takes as input a 64-bit data block and a 56-bit key. The 56-bit key is permuted into sixteen 48-bit subkeys, which are processed against the input data block. The output of the DES algorithm is another 64-bit data block. The difference between DES encryption and decryption is the order in which the subkeys are applied to the input data.

The application of the DES algorithm to encrypt discrete, 64-bit data blocks is called Electronic Code Block (ECB) mode. ECB is one of four DES modes of operation. These four DES modes of operation will be discussed later in this guide.

DES Keys

The 56-bit key is used by DES to generate a mapping between plaintext (unencrypted data) and ciphertext (encrypted data). DES is a symmetric cipher because same key is used for both encryption and decryption. DES

keys are usually stated as eight, 8-bit bytes. The least-significant bit of each byte of the key is discarded during the generation of the sixteen 48-bit subkeys. These bits are usually set so that each byte of the key has odd parity, known as “normal form”.

The security of the encrypted data is maintained only as long as the value of the key used to encrypt it is kept secret. In general, any randomly generated key should be as secure as any other key. In practice, though, there are some keys that are not desirable for use by DES. [6] identifies a number of DES keys that are considered to be either weak, semi-weak, or possibly weak. These keys are describe below.

Weak Keys

The DES keys, which are considered to be weak, are shown in Table 1. These keys are weak because the encrypted data generated by DES with these keys is identical to the unencrypted data. Note that there are only four weak keys out of 72,057,594,037,927,936 possible keys, so DES is not necessarily vulnerable due to the existence of weak keys. There are certainly other reasons that DES can become vulnerable, as will be discussed later. TclDES provides checks for the use of these keys during the generation of the subkeys, forcing an error if any of them are encountered.

Table 1: Weak DES Keys.

Key Value (hexidecimal with parity bits)	Actual Key
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 0E0E 0E0E	0000000 FFFFFFF
E0E0 E0E0 F1F1 F1F1	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

Semi-weak Keys

There are twelve DES keys that are considered semi-weak. What is meant by semi-weak in this case is that these keys form six pairs that are complement keys. A complement key exists for a given key if the bit-wise complement of that key encrypts the bit-wise complement of the plaintext into the bit-wise complement of the ciphertext. These complement key pairs are listed in Table Table 2. Rather than producing sixteen unique subkeys, the semi-weak keys produce sixteen subkeys that have only two unique values, each being used eight times in the DES algorithm. These keys should be avoided.

Table 2: Semi-weak DES Keys.

Key Value (hexidecimal with parity bits)	Complement Key Value (hexidecimal with parity bits)
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E0 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1

Possibly Weak Keys

There are also fourty-eight DES keys that a regarded as possibly weak. These keys produce only four distinct subkeys, rather than sixteen. The four subkeys are used four times in the DES algorithm. The fourty-eight possibly weak DES keys are listed in Table 3. These keys should be avoided.

Table 3: Possibly weak DES Keys.

Key Value (hexidecimal with parity bits)		
1F1F 0101 0E0E 0101	E001 01E0 F101 01F1	011F 1F01 010E 0E01
FE1F 01E0 FE0E 01F1	1F01 011F 0E01 010E	FE01 1FE0 FE01 0EF1
0101 1F1F 0101 0E0E	E01F 1FE0 F10E 0EF1	E0E0 0101 F1F1 0101
FE01 01FE FE01 01FE	FEFE 0101 FEFE 0101	E01F 01FE F10E 01FE
FEE0 1F01 FEF1 0E01	E001 1FFE F101 0EFE	E0FE 1F01 F1FE 0E01
FE1F 1FFE FE0E 0EFE	FEE0 011F FEF1 010E	1FFE 01E0 0EFE 01F1
E0FE 011F F1FE 010E	01FE 1FE0 01FE 0EF1	E0E0 1F1F F1F1 0E0E
1FE0 01FE 0EF1 01FE	FEFE 1F1F FEFE 0E0E	01E0 1FFE 01F1 0EFE
FE1F E001 FE0E F101	0101 E0E0 0101 F1F1	E01F FE01 F10E FE01
1F1F E0E0 0E0E F1F1	FE01 E01F FE01 F10E	1F01 FEE0 0E01 FEF1
E001 FE1F F101 FE0E	011F FEE0 010E FEF1	01E0 E001 01F1 F101
1F01 E0FE 0E01 F1FE	1FFE E001 0EFE F001	011F E0FE 010E F1FE
1FE0 FE01 0EF1 FE01	0101 FEFE 0101 FEFE	01FE FE01 01FE FE01
1F1F FEFE 0E0E FEFE	1FE0 E01F 0EF1 F10E	FEFE E0E0 FEFE F1F1
01FE E01F 01FE F10E	E0FE FEE0 F1FE FEF1	01E0 FE1F 01F1 FE0E
FEE0 E0FE FEF1 F1FE	1FFE FE1F 0EFE FE0E	E0E0 FEFE F1F1 FEFE

DES Modes Of Operation

Along with the DES algorithm, NBS/NIST also specified four modes of operation for the purpose of allowing DES to be used in a wide variety of applications. These modes of operation are: Electronic Code Block (ECB), Cipher Block Chaining (CBC), Output Feedback (OFB), and Cipher Feedback (CFB). The ECB and CBC modes of operation use the DES engine to process data in blocks of eight bytes each. OFB and CFB use the DES engine to process a stream of data bits. The modes are described in a separate FIPS publications, [7], from the DES algorithm, as they describe how the DES algorithm can be used rather than the algorithm itself. Each mode has advantages and disadvantages in their usage. These modes of operation are described in further detail below. A scholarly treatment of these modes of operation can be found in [8]. The illustrations used in describing the modes were taken from [9].

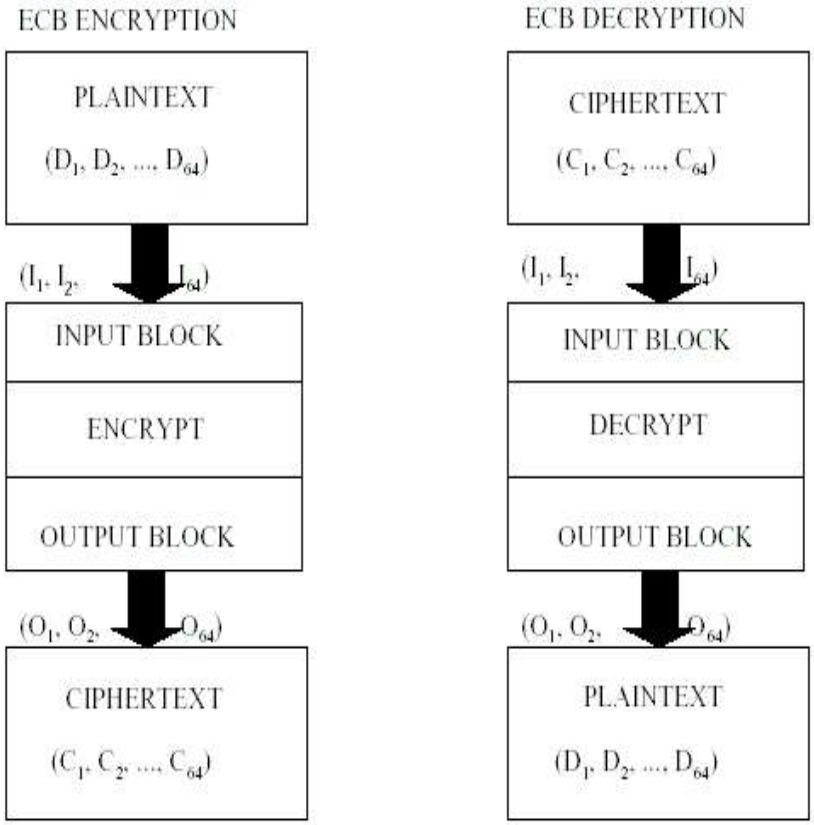

Illustration 1. Electronic Code Block (ECB) Mode Of Operation.

ECB Mode

Electronic Code Block (ECB) is the simplest DES mode of operation. A message is encrypted or decrypted in blocks of eight bytes each until all bytes of the message are processed. If the length of the plaintext message is not an integral multiple of eight bytes, then a sufficient number of dummy bytes (usually null characters) must be appended to complete the last eight-

byte block. A single 56-bit key is used to encrypt all of the message blocks. Illustration 1 shows how the DES algorithm is applied in the ECB mode of operation.

Beyond the simplicity of the ECB mode of operation, its other advantage is its relative robustness to corruption of the encrypted data. If one byte of encrypted data is corrupted, then the plaintext of only one eight-byte block is lost. All other blocks are unaffected by the corruption.

The primary problem with the ECB mode of operation is that it is the least secure application of the DES algorithm. Advances in computing technology have made it possible to break the 56-bit key of DES in a few days. This has been accomplished with high-speed, parallel computing, using a “brute-force” attack, as described in [4]. A “brute-force” attack means that each DES key, out of the possible 2^{56} keys, is applied to the ciphertext until one successfully decrypts the message. Consequently, the ECB mode of operation for DES is considered insecure for most applications.

To provide a practical level of security with DES, it is now recommended that one of the “Triple-DES” modes of operation be used to encrypt data. Triple-DES is described later in this guide.

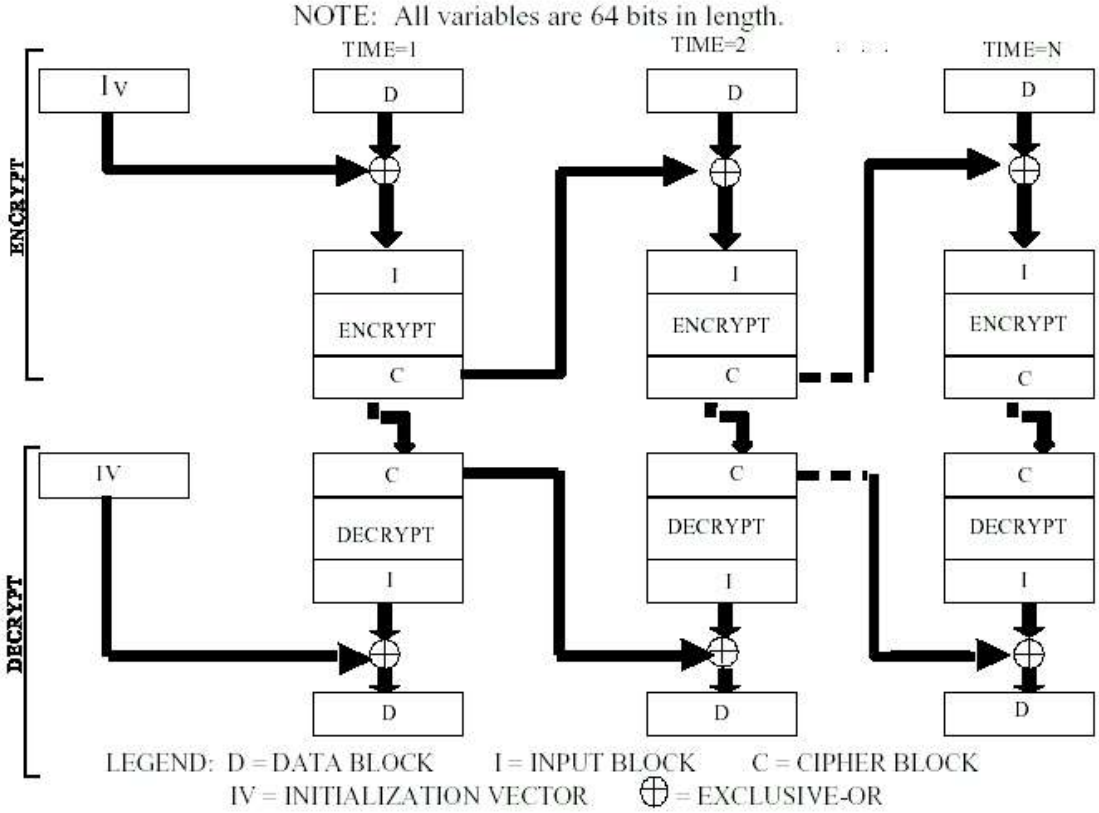

Illustration 2. Cipher Block Chaining (CBC) Mode Of Operation.

CBC Mode

The Cipher Block Chaining (CBC) mode of operation for DES is similar to ECB mode in that it encrypts and decrypts messages in eight-byte blocks. As with ECB mode, the message length must be an integral multiple of eight bytes or padded to obtain that length. CBC mode is different in that it uses the previously encrypted data block of the message as a mixing pattern for the current block of the message. During encryption, the previous block of ciphertext is fed forwand and exclusive-OR'ed with the current plaintext block of the message before encryption with the DES algorithm. During decryption, the previous block of ciphertext is fed forward and exclusive-OR'ed with the output of the current plaintext block of the message it has been decypted from ciphertext with the DES algorithm.

The first block of data to be encrypted or decrypted with CBC mode is a special case, as there is no previous block of ciphertext available to exclusive-OR with it. To resolve this problem, the user supplies a 64-bit initialization vector to act as a mixing pattern for the first block of message data. The CBC mode of operation is shown in Illustration 2. Note that if it is desired to encrypt or decrypt a message as several sub-messages, all that is necessary to do this is that the last eight-byte block of ciphertext be retained to act as the initialization vector for the next sub-message. For this to work, each sub-message must be an integral multiple of eight bytes.

CBC mode is a little more complicated to implement than ECB mode. It is required that a block of ciphertext be carryed over from one iteration of the encryption or decryption activity to the next. The initialization vector must also be generated and communicated to those who need to decrypt the message.

The robustness of CBC mode to data corruption is less than that of ECB mode. If a byte of the ciphertext message is corrupted, that block and all subsequent blocks of the plaintext message cannot be recovered during decryption. Consequently, messages encrypted using CBC mode should be transported using a reliable storage media or communication channel.

The extra effort required by CBC mode buys the user greater security over ECB mode. In order to read the plaintext message, an adversary has to know both the key and the 64-bit initialization vector used encrypt it. Even if the key happened to be guessed through a brute-force attack, each plaintext block is mixed with either the initialization vector or a ciphertext block whose value is ultimately based on the initialization vector. Of course, if the key and initialization vector for the first eight-byte block of ciphertext are discovered, then all subsequent blocks can be easily decrypted. A brute-force attack of both the key and the initialization vector would require a maximum of $2^{56} * 2^{64}$ or 2^{110} guesses.

OFB Mode

The Output Feedback (OFB) mode of operation provides a means by which DES can be used as stream cipher. This means that the encryption algorithm processes the message as a sequence of bits. The messages is segmented into K-bit chunks, where K may equal 1 through 64, inclusively. To start an encryption or decryption, an initialization vector of length L is supplied to the input of the DES encryption algorithm. The value of L may equal 1 through 64, inclusively, but does not have to equal the K-bit segment size. The initialization vector is placed in the least-significant bits of the input. The remaining 64-L bits are set to zero.

The most-significant K bits of the output of the DES encryption algorithm are mixed with K bits of the message bit stream using an exclusive-OR operator. All 64 bits of the output are also fed back into the input of the DES encryption algorithm to be encrypted again. The DES encryption algorithm is used as a pseudo-random bit pattern generator. The OFB mode of operation is shown in Illustration 3.

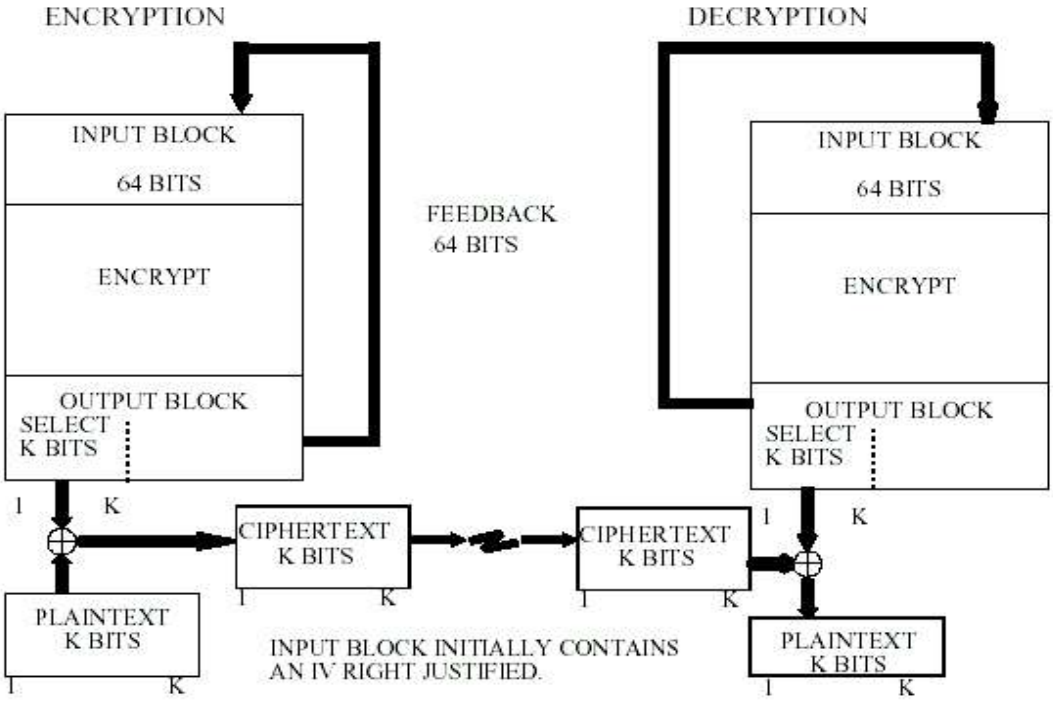

Illustration 3. Output Feedback (OFB) Mode Of Operation.

OFB mode is relatively simple to implement. The DES encryption algorithm is used in the same fashion as ECB mode with the output fed back into the input. The additional processing is in taking the K bits of the encryptor output and mixing them with the message bit stream. In practice, the message data is usually not available as a true bit stream, but as a sequence of bytes. Therefore, unpacking and repacking of bits must occur if all bits in every byte of the message are to contain actual data. Note that the OFB mode function in the OpenSSL library does not use packed binary byte streams.

Rather, each K-bits of data are stored in the smallest integral multiple of bytes that will hold the K bits. Those bits that are not part of the K-bits of the message are set to zero.

The robustness of OFB mode to data corruption is good. If any bit of the ciphertext message is corrupted, only that bit is affected in the decryption. This is due to the independence between bits in the message stream. On the other hand, synchronization must be maintained between the DES encryptors at the transmitting and receiveing ends of the ciphertext message. If the DES encryptors become unsynchronized, then none of the plaintext can be recovered. Consequently, both encryptors must be restarted.

A ciphertext bit stream encrypted using OFB mode is vulnerable to interception if an adversary has access to the output of the DES encrypted ciphertext sequence used to mix with the plaintext bit stream. The encryption key and initialization vector are not needed. Of course, the adversary's DES encrypted ciphertext sequence must be synchronized with the ciphertext bit stream in order to recover the plaintext bit stream. A brute-force attack to guess the DES key value, the initialization vector value, and the size of K would require a maximum of $2^5 * 2^{56} * 2^{64}$ or 2^{115} guesses.

CFB Mode

The Cipher Feedback (CFB) mode of operation uses the DES encryption algorithm to implement a stream cipher, like OFB mode. CFB mode is slightly more complicated, though. The messages is segmented into K-bit chunks, where K may equal 1 through 64, inclusively. To start an encryption or decryption, an initialization vector, also of length K bits, is supplied to the input of the DES encryption algorithm. The initialization vector is placed in the least-significant bits of the input. The remaining 64-K bits are set to zero.

With CFB encryption, the most-significant K bits of the output of the DES encryption algorithm are mixed with K-bit segment of the plaintext bit stream using an exclusive-OR operator. This forms a K-bit ciphertext bit stream. The previous input of the DES encryption algorithm is shifted to the left by K bits and the K-bit segment of the ciphertext bit stream that were just created are inserted into the least-significant K bits of the input. The DES encryption algorithm is ready for the next iteration of the pseudo-random bit pattern generator for CFB encryption.

With CFB decryption, the most-significant K bits of the output of the DES encryption algorithm are mixed with K bits of the ciphertext bit stream using an exclusive-OR operator. This forms a K-bit plaintext bit stream. The previous input of the DES encryption algorithm is shifted to the left by K bits and the same K bits of the ciphertext bit stream are inserted into the least-significant K bits of the input. This prepares the DES encryption algorithm for the next iteration of the pseudo-random bit pattern generator for CFB mode decryption. The OFB mode of operation is shown in Illustration 4.

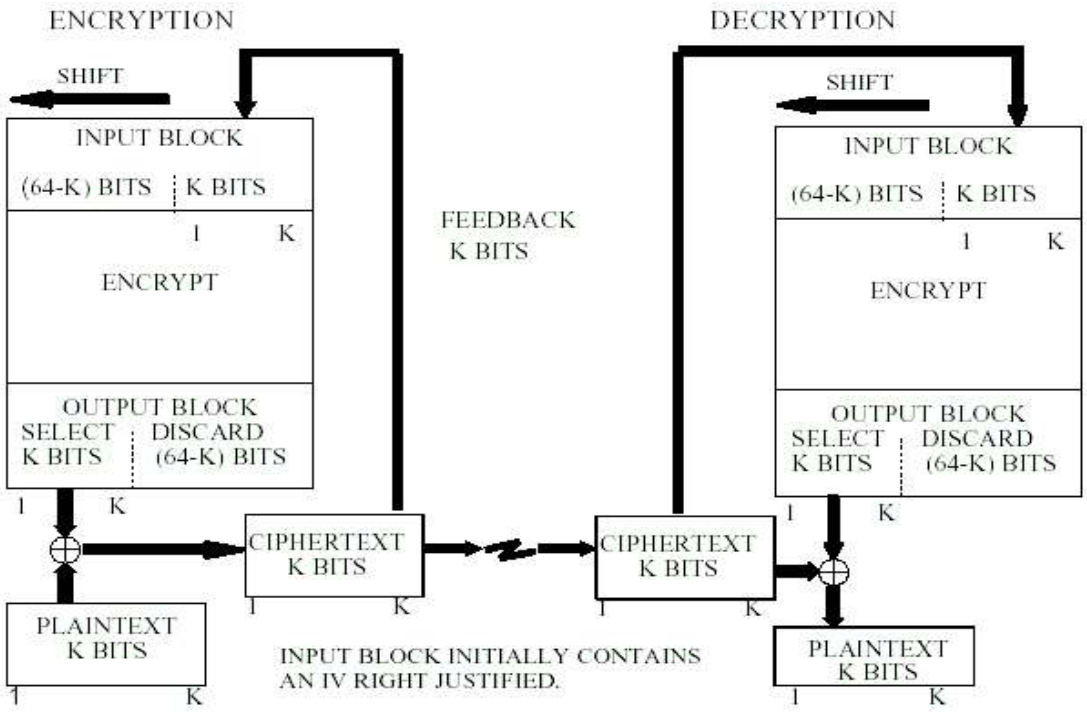

Illustration 4. Cipher Feedback (CFB) Mode Of Operation.

CFB mode is a bit more complicated to implement as compared to OFB mode. The DES encryption algorithm is used in the same fashion as ECB mode. The additional processing is in taking the K bits of the encryptor output and mixing them with the message bit stream. In addition it is also necessary to shift the K-bit segment of the input to the DES encryption algorithm and inserting the K bits of the cipher bit stream. In practice, the message data is usually not available as a true bit stream, but as a sequence of bytes. Therefore, unpacking and repacking of bits must occur if all bits in every byte of the message are to contain actual data. Note that the CFB mode function in the OpenSSL library does not use packed binary byte streams. Rather, each K-bits of data are stored in the smallest integral multiple of bytes that will hold the K bits. Those bits that are not part of the K-bits of the message are set to zero.

The robustness of CFB mode to data corruption is comparable to CBC mode. If any bit of the ciphertext message is corrupted, all bits in that K-bit block and subsequent blocks of the plaintext message cannot be recovered during decryption. Consequently, messages encrypted using CFB mode should be transported using a reliable storage media or communication channel. In addition, synchronization must be maintained between the DES encryptors at the transmitting and receiveing ends of the ciphertext message. If the DES encryptors become unsynchronized, then none of the plaintext can be recovered. Consequently, both encryptors must be restarted.

A ciphertext bit stream encrypted using CFB mode is much more secure than OFB mode. This is due to the input of the ciphertext bit stream into the DES

encryptor. In order to read the plaintext message, an adversary has to know both the key and the K-bit initialization vector used encrypt it. Of course, the adversary's DES encrypted ciphertext sequence must be synchronized with the ciphertext bit stream in order to recover the plaintext bit stream. A brute-force attack would have to guess the DES key value, the initialization vector value, and the size of K. A brute-force attack of the key, the initialization vector, and the value of K would require a maximum of $2^5 * 2^{56} * 2^{64}$ or 2^{115} guesses.

Weak Versus Strong Encryption: Triple-DES

As mentioned previously, advances in technology have made DES vulnerable to attack due to the relatively short length of it key. DES, especially DES-ECB mode of operation, is considered weak encryption. In order to provide additional security, an extended version of DES called Triple-DES (3DES), or Triple Data Encryption Algorithm (TDEA), was added to the standard. Triple-DES applies the standard DES algorithm three times successively to each block of data. 3DES is currently regarded as strong encryption.

Encryption using 3DES is performed using an encryption-decryption-encryption (EDE) sequence. The first application of the DES algorithm is an encryption operation on the 64-bit plaintext block. The second application of the DES algorithm is a decryption operation on the output of the encrypted 64-bit block. The third application of the DES algorithm is another encryption operation on the decrypted 64-bit block. Decryption using 3DES is performed using an decryption-encryption-decryption sequence. The first application of the DES algorithm is an decryption operation on the 64-bit ciphertext block. The second application of the DES algorithm is an encryption operation on the output of the decrypted 64-bit block. The third application of the DES algorithm is another decryption operation on the encrypted 64-bit block.

All of this activity seems rather pointless if the 56-bit keys for each application of the DES algorithm were the same. If they were, then 3DES encryption would be no different than a single application of the DES algorithm. With 3DES, each 56-bit key can be different, but they don't have to be. In fact, one approved mode of 3DES uses the same key for each application of the DES algorithm to maintain compatibility with standard DES. Of course, this usage of 3DES is just as susceptable to compromise as standard DES. The more secure methods of using 3DES involves either two or three different 56-bit keys.

When two different keys (K_1 and K_2) are used, this form of 3DES is called DES-EDE2. During DES-EDE2 encryption, key K_1 is used for the two DES encryption stages and key K_2 is used for the intermediate DES decryption stage. During DES-EDE2 encryption, key K_1 is used for the two DES decryption stages and key K_2 is used for the intermediate DES encryption stage. DES-EDE2 is much more secure than standard DES. A brute-force

attack to determine both 56-bit keys would take $2^{56} * 2^{56}$ or 2^{112} guesses.

When three different keys (K_1, K_2, and K_3) are used, this form of 3DES is called DES-EDE3. During DES-EDE3 encryption, key K_1 is used for the first DES encryption stage, key K_2 is used for the intermediate DES decryption stage, and K_3 is used for the last DES encryption stage. During DES-EDE3 decryption, key K_3 is used for the first DES decryption stage, key K_2 is used for the intermediate DES encryption stage, and K_1 is used for the last DES decryption stage. DES-EDE3 is extreemly secure when compared to standard DES. A brute-force attack to determine both 56-bit keys would take $2^{56} * 2^{56} * 2^{56}$ or 2^{168} guesses. Brute-force attacks against both DES-EDE2 and DES-EDE3 are considered impractical.

Just as with standard DES, NIST has specified several modes of opereration for using the 3DES algorithm:

- Triple DES-ECB (3DES-ECB) also known as TDEA-ECB (TECB)
- Triple DES-CBC (3DES-CBC) also known as TDEA-CBC (TCBC)
- Triple DES-OFB (3DES-OFB) also known as TDEA-OFB (TOFB)
- Triple DES-CFB (3DES-CFB) also known as TDEA-CFB (TCFB)

These modes of operation are identical to those that use the standard DES algorithm, except the 3DES algorithm is used instead. Of course, either two or three 56-bit keys need to be specified.

Using TclDES

The TclDES library package implements the DES and 3DES algorithms and all modes of operation in pure Tcl. No platform-specific extensions are required. Support functions to generate and manage key sets are also provided.

Since 3DES is regarded as strong encryption, the legality of transferring the TclDES library across international borders must be taken into consideration. Consequently, a reduced-functionality version of TclDES exists with the 3DES support removed. This library is called TclDESjr. For most of the proceeding discussion, either library can be used interchangably. The examples that use 3DES or long key set generation only apply to TclDES, not TclDESjr.

TclDES Installation

To install the TclDES or TclDESjr library, first extract the distribution from the archive file. If the distribution file is a UNIX tar file compressed with gzip, enter the following commands from within the directory where the distribution will go:

TclDES	TclDESjr
`gzip -d tclDES-0.8.tar.gz`	`gzip -d tclDESjr-0.8.tar.gz`
`tar xvf tclDES-0.8.tar`	`tar xvf tclDESjr-0.8.tar`

or if GNU tar is available, simply enter:

TclDES	TclDESjr
`tar xzvf tclDES-0.8.tar.gz`	`tar xzvf tclDESjr-0.8.tar.gz`

If the distribution file is a PKZIP file, enter the following commands from within the directory where the distribution will go:

TclDES	TclDESjr
`unzip tclDES-0.8.zip` for Unix	`unzip tclDESjr-0.8.zip` for Unix
`pkzip tclDES-0.8.zip` for DOS	`pkzip tclDESjr-0.8.zip` for DOS

or use WINZIP under Microsoft Windows.

The distribution, extracted from the archive file, should contain the following:

TclDES	TclDESjr
`tclDES-0.8/readme.txt`	`tclDESjr-0.8/readme.txt`
`tclDES-0.8/tcldes.html`	`tclDESjr-0.8/tcldesjr.html`
`tclDES-0.8/tcldes.n`	`tclDESjr-0.8/tcldesjr.n`
`tclDES-0.8/AGuideToTclDES.pdf`	`tclDESjr-0.8/AGuideToTclDES.pdf`
`tclDES-0.8/tclDES0.8/des.tcl`	`tclDESjr-0.8/tclDESjr0.8/desjr.tcl`
`tclDES-0.8/tclDES0.8/pkgIndex.tcl`	`tclDESjr-0.8/tclDESjr0.8/pkgIndex.tcl`

The library directory tclDES0.8 or tclDESjr0.8 and its contents are then copied into a directory whose path is listed in the Tcl `auto_path` variable. Alternately, the path to the TclDES or TclDESjr library can be added to the `auto_path` variable from within the Tcl interpreter.

Loading TclDES

The TclDES or TclDESjr package is loaded into the Tcl interpreter by using the `package require` command:

TclDES	TclDESjr
`package require tclDES <0.8>`	`package require tclDESjr <0.8>`

Optionally, the version of the library can be specifed if several versions of the library are available to the Tcl interpreter.

Once loaded, there should now be a `::des` namespace avalible to the Tcl interpreter. This namespace contains the commands for the TclDES or TclDESjr library. Note that either the TclDES or the TclDESjr library should be loaded into a Tcl interpreter, but not both. Both librarys use the same

command set, so they cannot exist in the same namespace.

Key Set And Initialization Vector Management

As discussed previously, the DES algorithm uses a 56-bit key, which is specified as a sequence of eight bytes. The least-significant bit of each byte is unused by DES. When the DES key is presented in “normal form”, these bits are are used to provide odd parity for their respective bytes. TclDES does not require that the bytes of the key have odd parity, though, in order for the key to work properly.

A DES key can be created via any number of means, either automated or manually. The key must be a eight-byte binary string, though an eight-character ASCII string can be used. A binary string provides a must larger set of possible strings and allows the key to be expressed in normal form if desired. Binary strings provide a higher measure of security, as they are harder to guess than ASCII strings. Below is an example of generating a DES key using the Tcl `rand` function:

```
# Generate a 64-bit binary key from four 16-bit fragments.
set fragmentA [format %04x [expr int(65536 * rand())]]
set fragmentB [format %04x [expr int(65536 * rand())]]
set fragmentC [format %04x [expr int(65536 * rand())]]
set fragmentD [format %04x [expr int(65536 * rand())]]
set key [binary format H4H4H4H4 $fragmentA $fragmentB $fragmentC $fragmentD]
```

Note that the Tcl `rand` function is not cryptographically secure, as it uses a pseudo-random number generator to create the values. Before generating keys, it is recommended to review the information provided in the section **DES Keys**.

Before a key can be used by the DES algorithm, it must be processed to form the sixteen 48-bit subkeys that are applied to the data. The processing used to create the subkeys only has to be done once. The subkeys can be used multiple times by the DES algorithm for both encryption and decryption.

TclDES provides a key set generation and management facility to streamline the handling of the DES subkeys. Rather than passing keys or subkeys to the DES encryption and decryption functions, a handle to a key set is passed instead. The handle is used as an index into the Tcl array, `des::keysets`, which resides within the ::`des` namespace. Each array element consists of a list of sixteen 48-bit subkeys that have been generated previously.

The `des::keyset` command is the user interface to the keyset facility. To create a new key set, the `create` option is used, with the key value being supplied as a parameter by the user. The key set is created and placed into the `des::keysets` array. The index to the key set is automatically assigned and has the form `keysetN`, where `N` is a the value of the array variable `des::keysets(ndx)`. The value of `des::keysets(ndx)` is set to 1 when the

TclDES library is loaded and is incremented by 1 after each new key set is created. A key set handle value only be issued once and will never be reissued until the TclDES library is reinstalled.

When an eight-byte string, representing a 56-bit key, is supplied a key set for standard DES is created:

```
% set ks_handle1 [des::keyset create ABCDEFGH]
keyset1
% puts $des::keysets($ks_handle1)
151138818 671679530 17957383 671679500 84942852 151596290 621806083 17109007
605104649 50662656 873013506 50599957 805898794 33948420 269026560 102897969
268970240 102896161 34343992 69349396 33820936 70255160 35915056 271595008
170136620 806354984 136446736 806888482 153227317 940058632 153236480
671623178
```

Note that each 48-bit subkey is represented as two 32-bit values due to the implementation of the DES algorithm. When a twenty-four byte string, representing a 168-bit key (three, 56-bit keys), is supplied a key set for 3DES is created:

```
% set ks_handle2 [des::keyset create ABCDEFGHIJKLMNOPQRSTUVWX]
keyset2
% puts $des::keysets($ks_handle2)
151138818 671679530 17957383 671679500 84942852 151596290 621806083 17109007
605104649 50662656 873013506 50599957 805898794 33948420 269026560 102897969
268970240 102896161 34343992 69349396 33820936 70255160 35915056 271595008
170136620 806354984 136446736 806888482 153227317 940058632 153236480
671623178 151142186 738788890 17958703 673777212 118498612 151588642
621811251 18165799 605108761 319105320 1007235350 50609181 806033454 33949454
269027077 237114675 285739781 102899235 34352699 69416978 100938507 70257468
35923250 271855365 170204198 806359341 404883992 806892851 153229597
940194329 153237800 671627802 151663106 738923819 17959431 741020685
118499340 220806418 655622923 18158111 605368105 320147200 1009329442
319035445 942344762 570827556 269161744 774519097 285882624 237646377
51121212 203633692 101978380 70321722 104073009 288634370 707081773 823396392
941826834 840444966 422187063 973744396 422196224 671754511
```

As before, each 48-bit subkey is represented as two 32-bit values. Note that the `create` option in the TclDESjr version of the `des::keyset` command only supports the creation of subkeys for 56-bit DES keys.

Once the need for a key set has passed, it may be removed from the `des::keysets` array by using the `destroy` option of the `des::keyset` command, supplying the handle name as a parameter:

```
% des::keyset destroy keyset1
```

The subkey generation facility can be accessed directly through the `des::createKeys` command. The only parameter required is an eight-byte string, representing a 56-bit key, or a twenty-four byte string, representing a 168-bit key (three, 56-bit keys). A list containing the subkeys, as described above, is returned. Note that subkey lists created in this fashion cannot be used by the encryption and decryption functions of TclDES, as they are not

stored in the `des::keysets` array in the ::des namespace.

The CBC, OFB, and CFB modes of operation require an initialization vector to provide an initial feed-forward value (for CBC mode) or feedback value (for OFB and CFB modes). Like the DES key, the initialization vector is represented as a eight-byte string, preferably binary. TclDES uses the variable that stores the initialization vector as a placeholder of the resulting feed-forward or feedback data that is available at the end of the final processing cycle of the CBC, OFB, and CFB modes modes of the DES and 3DES algorithms.

In TclDES, the initialization vector is passed via reference into the encryption or decryption procedure. That is, the name of the variable containing the initialization vector is passed, not the value of the vector. After the procedure has fininshed execution, the original content of the variable is replaced by the feed-forward or feedback value. The advantage of this approach is that the encryption or decryption of a long message can be spread across multiple calls of the procedure rather than all at once. This is very useful when coupled with event-driven processing. The disadvantage is that the original value of the initialization vector is lost. Consequently, a copy of any initialization vector used for CBC, OFB, or CFB encryption must be retained elsewhere so that the message can be decrypted later using the same initialization vector.

TclDES Encryption Comands and Examples

TclDES provides support for all four DES/3DES modes of operation. There are four commands in the `des` namespace that supports two ways for the modes can be accessed. These commands will be described briefly below. Examples of how the commands are used are presented afterward. Details on these commands and their parameters can be found in the TclDES and TclDESjr manual pages.

The first way that the DES/3DES modes are accessed is through the `des::encrypt` and `des::decrypt` commands. These commands provide a consistent interface to all four modes of operation and are oriented according to whether encryption or decryption is being performed. Both commands require as input parameters a key set handle and the message data. All other parameters are optional. These parameters specify the mode of operation (ECB by default), the name of the variable containing the initialization vector used by the CBC, OFB, and CFB modes (an empty string by default), and the bit segment size used by the OFB and CFB modes (64 bits by default). If any mode other than ECB are specified, then the name of the variable containing the initialization vector must be specified. If the CBC mode is specified, then the bit segment size value is ignored. The output of the `des::encrypt` and `des::decrypt` commands is the encrypted or decrypted message, respectively.

The `des::encrypt` and `des::decrypt` commands are wrappers around the `des::block` and `des::stream` commands. These commands represent the other way to access the DES/3DES modes of operation. Using these commands directly removes a layer of procedure call, but require that the user be familiar with the appropriate values for the parameters, which are a bit more cryptic.

The `des::block` command supports the ECB and CBC modes of operation. The command requires as input parameters a key set handle, the message data, and an encryption/decryption flag. Optional parameters are a flag representing the mode of operation ('0' represents ECB mode, which is the default) and the name of the variable containing the initialization vector (an empty string by default). If the mode flag indicates CBC mode ('1' represents CBC mode), then the variable containing the initialization vector must be specified.

The `des::stream` command supports the OFB and CFB modes of operation. The comand requires as input parameters a key set handle, the message data, and an encryption/decryption flag. In addition, a mode flag ('0' represents OFB mode and '1' represents CFB mode) and the name of the variable containing the initialization vector are required. An optional parameter is the bit segment size used by these modes (64 bits by default).

While the stream ciphers are designed to encrypt and decrypt continuous sequences of bits, the most common format in which bits are conveyed is 8-bit bytes. The value of the bit segment size, though, can range from 1 to 64 bits. If the bit segment size is an integral multiple of eight bits, then the bit stream can be easily processed through the OFB and CFB algorithms as bytes. If this is not the case, then unpacking and repacking of bits from bytes is required to process the appropriate number of bits during each algorithm cycle. Consequently, if the length of the message in bits is not an integral multiple of the bit segment size, then there will not be an sufficient number of bits available for the last algorithm cycle. Conversly, if the message does not fit within an integral number of bytes, then the last byte must contain unused bits and some mechanism must be created to indicate which bits are not part of the actual message.

DES OFB and CFB mode functions in the OpenSSL library solve this problem by not using packed binary byte streams. Rather, the message stream is broken ino chunks of data that are equal to the length of the bit segment size. Each chunk of data is stored in the smallest integral multiple of bytes that will hold the bits. Those bits that are not part of the bit segment are not used and are set to zero. TclDES uses packed bit data, so no bits are wasted in any byte. As a result, the length of the message must be an integral multiple of a string of bytes sufficient to satisfy the conditions described above. Table 4 lists the integral message length required for each bit segment size.

Table 4: Required Integral Message Length For Bit Segment Size.

Bit Segment Size (bits)	Integral Message Length (bytes)	Bit Segment Size (bits)	Integral Message Length (bytes)	Bit Segment Size (bits)	Integral Message Length (bytes)
1	1	23	69	45	270
2	1	24	3	46	276
3	3	25	100	47	282
4	1	26	104	48	6
5	5	27	108	49	343
6	6	28	112	50	350
7	7	29	116	51	357
8	1	30	120	52	364
9	18	31	124	53	371
10	20	32	4	54	378
11	22	33	165	55	385
12	24	34	170	56	7
13	26	35	175	57	456
14	28	36	180	58	464
15	30	37	185	59	472
16	2	38	190	60	480
17	51	39	195	61	488
18	54	40	5	62	496
19	57	41	246	63	504
20	60	42	252	64	8
21	63	43	258		
22	66	44	264		

Block Cipher Examples

Below are several examples of how the DES/3DES block cipher modes of operation, ECB and CBC, are invoked. Note that the size of the key used to create the key set determines whether a particular encryption or decryption operation is stardard DES or 3DES. Of course, 3DES encryption and decryption are not available with the TclDESjr library.

ECB Mode

Given the variable `keyset`, which contains a handle to a set of subkeys, and the variable `data`, which contains a plaintext message, the ECB mode of operation for encrypt and decryption can be performed as follows:

```
set ct [des::encrypt $keyset $data ecb]
set pt [des::decrypt $keyset $ct ecb]
```

The default mode of operation for `des::encrypt` and `des::decrypt` is ECB, so the mode option can be unspecified if ECB mode is desired:

```
set ct [des::encrypt $keyset $data]
set pt [des::decrypt $keyset $ct]
```

The `des::block` command can be used in a similar fashion:

```
set ct [des::block $keyset $data 1 0]
set pt [des::block $keyset $ct 0 0]
```

CBC Mode

The initialization vector used in CBC mode is passed via reference rather than by value. This allows the variable to assume the value of the last feed-forward ciphertext block at the completion to the command. The initialization vector used for encryption must also be used for decryption, so a copy must be retained. The CBC mode of operation for encrypt and decryption can be performed as follows:

```
set ivec2 $ivec1
set ct1 [des::encrypt $keyset $data1 cbc ivec1]
set ct2 [des::encrypt $keyset $data2 cbc ivec1]
set pt1 [des::decrypt $keyset $ct1 cbc ivec2]
set pt2 [des::decrypt $keyset $ct2 cbc ivec2]
```

The `des::block` command can be used in a similar fashion:

```
set ivec2 $ivec1
set ct1 [des::block $keyset $data1 1 1 ivec1]
set ct2 [des::block $keyset $data2 1 1 ivec1]
set pt1 [des::block $keyset $ct1 0 1 ivec2]
set pt2 [des::block $keyset $ct2 0 1 ivec2]
```

Stream Cipher Examples

Below are several examples of how the DES/3DES stream cipher modes of operation, OFB and CFB, are invoked. Note that the size of the key used to create the key set determines whether a particular encryption or decryption operation is stardard DES or 3DES. Of course, 3DES encryption and decryption are not available with the TclDESjr library.

OFB Mode

Let the variable `keyset` contain a handle to a set of subkeys and the variable

`data` contain a plaintext message. The initialization vector used in OFB mode is passed via reference rather than by value. This allows the variable to assume the value of the last feedback block at the completion to the command. The initialization vector used for encryption must also be used for decryption, so a copy must be retained. The bit segment size (in this example a value of '23') is the final parameter. The OFB mode of operation for encrypt and decryption can be performed as follows:

```
set ivec2 $ivec1
set ct1 [des::encrypt $keyset $data1 ofb ivec1 23]
set ct2 [des::encrypt $keyset $data2 ofb ivec1 23]
set pt1 [des::decrypt $keyset $ct1 ofb ivec2 23]
set pt2 [des::decrypt $keyset $ct2 ofb ivec2 23]
```

Note that the length of each message must an integral multiple of the minimum message size required by the specified bit segment size (for this example, it would be 69 bytes). For a default bit segment size of 64 bits, the last parameter can be unspecified:

```
set ivec2 $ivec1
set ct1 [des::encrypt $keyset $data1 ofb ivec1]
set pt1 [des::decrypt $keyset $ct1 ofb ivec2]
```

The `des::stream` command can be used in a similar fashion:

```
set ivec2 $ivec1
set ct1 [des::stream $keyset $data1 1 0 ivec1 23]
set ct2 [des::stream $keyset $data2 1 0 ivec1 23]
set pt1 [des::stream $keyset $ct1 0 0 ivec2 23]
set pt2 [des::stream $keyset $ct2 0 0 ivec2 23]
```

CFB Mode

The input parameters for the CFB mode of operation are identical to those of the OFB mode of operation except for the mode selection parameter. for encrypt and decryption can be performed as follows:

```
set ivec2 $ivec1
set ct1 [des::encrypt $keyset $data1 cfb ivec1 56]
set ct2 [des::encrypt $keyset $data2 cfb ivec1 56]
set pt1 [des::decrypt $keyset $ct1 cfb ivec2 56]
set pt2 [des::decrypt $keyset $ct2 cfb ivec2 56]
```

As with OFB mode, the length of each message must an integral multiple of the minimum message size required by the specified bit segment size (for a size of 56 bits, it would be 7 bytes). For a default bit segment size of 64 bits, the last parameter can be unspecified:

```
set ivec2 $ivec1
set ct1 [des::encrypt $keyset $data1 cfb ivec1]
set pt1 [des::decrypt $keyset $ct1 cfb ivec2]
```

The `des::stream` command can be used in a similar fashion:

```
set ivec2 $ivec1
```

```
set ct1 [des::stream $keyset $data1 1 1 ivec1 56]
set ct2 [des::stream $keyset $data2 1 1 ivec1 56]
set pt1 [des::stream $keyset $ct1 0 1 ivec2 56]
set pt2 [des::stream $keyset $ct2 0 1 ivec2 56]
```

Import And Export Issues

Why is TclDES a munition?

Under the International Traffic in Arms Regulations (ITARs), encryption software and hardware are considered munitions along with guns, tanks, nuclear, biological, and chemical weapons. Encryption can potentially be used by adversaries to conceal their communications from the United States government. As a result, export of encryption software and hardware requires licensing and approval by the U.S. government.

Fortunately, export restrictions have been eased significantly in recent years. This is due to several reasons. For one, the large amount of commerce now occuring on the Internet requires that encryption be widely available. With commerce equating to money, there is a strong desire by the U.S. government for American companies to remain competitive. Another reason is that it has become virtually impossible to control the flow of cryptographic software into and out of the United States. Many books on encryption, containing source code, are readily available. Ironically, these books have no restrictions on export due to the 1st Ammendment of the U.S. Constitution. Also, the availablility of many encryption packages (such as OpenSSL and PGP) make the entire effort of controlling encryption export rather moot.

How is TclDES being made available then?

The TclDES source code is being made publicly available and has been registered with the U.S. Dept. of Commerce Bureau of Industry and Security (BIS) under export license exception TSU (Technology and Software Unrestricted) for export out of the United States. To qualify for this exception, the sources to TclDES must be made available with minimial or no cost. To satisfy this requirement, the TclDES sources are licensed under the same Open Source license as Tcl (BSD). Details can be found in the TclDES source code.

While license exception TSU allows for the export of TclDES out of the United States, the import of the strong encryption (3DES) contained within TclDES into other countries could be an issue. The BIS Export Administration Regulations (EARs) and the Wassenaar Arrangement allow for the unrestricted export (no licensing required) of symmetric key encryption with key lengths of 56 or fewer bits. The Wassenaar Arrangement is signed by 33 founding countries, which includes most of the major industrialized nations. To satisfy this restriction and make a pure-Tcl implementation of DES as widely available as possible, a version of the TclDES source code was created

with the 3DES capabilities stripped out. This version is called TclDESjr.

Regardless of the statements made above, it is incumbent upon the user to determine the legalities of the use of TclDES or TclDESjr in their locale or the locale of their customer. The author of TclDES and TclDESjr is not liable for the inappropriate import or export of either library by others.

Glossary

Below are the definitions of a number of terms used in the discussion of encryption and decryption

block cipher	An encryption algorithm that operates on discrete blocks of data of a fixed size, usually along byte boundaries.
ciphertext	Commonly, text which has been encrypted or made unintelligible using an encryption algorithm, though generally referring to any binary data that has been encrypted.
complement key	A complement key exists for a given key if the bit-wise complement of that key encrypts the bit-wise complement of the plaintext into the bit-wise complement of the ciphertext.
initialization vector	A 64-bit block of data used by DES CBC, CFB, and OFB modes of operation to supply the initial feedback data.
key	A data pattern used by a cipher to encrypt or decrypt data.
normal form	The formatting of an eight-byte DES key, where the least-significant bit of each byte is set so that the bytes have odd parity.
plaintext	Commonly, text which which can be directly understood by any observer, though generally referring to any binary data that can be directly understood.
possibly-weak key	A DES key that produces only four unique subkeys rather than the normal unique sixteen subkeys.
semi-weak key	A DES key that forms a member of a complement key pair.
stream cipher	An encryption algorithm that operates on serial data, usually a sequence of bits.

symmetric cipher	A cipher in which the same key is used for both encryption and decryption.
weak key	A DES key by which the encrypted data generated by the DES algorithm is identical to the unencrypted data.

References

[1] Federal Information Processing Standards Publication, FIPS PUB 46-3, DATA ENCRYPTION STANDARD (DES), U.S. Department Of Commerce/National Institute of Standards and Technology, October 25, 1999 (http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf).

[2] Wikipedia, the free encyclopedia, Data Encryption Standard, http://en.wikipedia.org/wiki/Data_Encryption_Standard).

[3] RSA Laboratories' Frequently Asked Questions About Today's Cryptography. Version 4.1, RSA Laboratories, 20 Crosby Drive Bedford, MA 01730 USA , (ftp://ftp.rsasecurity.com/pub/labsfaq/rsalabs_faq41.pdf).

[4] Grabbe, J. Orlin, The DES Algorithm Illustrated, (http://www.aci.net/kalliste/des.htm).

[5] Schneier, Bruce, Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C, Wiley Computer Publishing, John Wiley & Sons, Inc.

[6] Federal Information Processing Standards Publication, FIPS PUB 74, GUIDELINES FOR IMPLEMENTING AND USING THE NBS DATA ENCRYPTION STANDARD, U.S. Department Of Commerce/National Institute of Standards and Technology, April, 1981 (http://www.itl.nist.gov/fipspubs/fip74.htm).

[7] Federal Information Processing Standards Publication, FIPS PUB 81, DES MODES OF OPERATION, U.S. Department Of Commerce/National Institute of Standards and Technology, December 2, 1980 (http://www.itl.nist.gov/fipspubs/fip81.htm).

[8] Handbook of Applied Cryptography, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996 (http://www.cacr.math.uwaterloo.ca/hac).

[9] National Institute of Standards and Technology, NIST Special Publication 800-17, Modes of Operation Validation System (MOVS): Requirements and Procedures.