

Chapter 30

Encryption Services for x900-24X Switches

Introduction	30-2
Data Encryption	30-2
Symmetrical Encryption	30-3
Asymmetrical (Public Key) Encryption	30-4
Payload Encryption	30-4
Authentication	30-5
Key Exchange Algorithms	30-6
ENCO Services	30-7
Encryption	30-7
Authentication	30-8
Diffie-Hellman Key Exchange Algorithm	30-8
Key Creation and Storage	30-8
Access Control	30-10
Command Reference	30-11
create enco key	30-11
destroy enco key	30-13
disable enco compstatistics	30-14
disable enco debugging	30-14
enable enco compstatistics	30-14
enable enco debugging	30-15
reset enco counter	30-16
set enco dhpriority	30-16
set enco key	30-17
show enco	30-18
show enco channel	30-19
show enco counter	30-24
show enco debug	30-35
show enco key	30-35

Introduction

This chapter describes the data security services available on the AT-9924Ts and x900-24X switches, how the services are provided, the switch network functions that use these services, and how to monitor the services.

A higher layer module must be configured to use encryption. Encryption is currently supported on Secure Shell (SSH) and Secure Sockets Layer (SSL). The ENCO module provides data encryption and key creation and storage to these protocols. For more information, see [Chapter 33, Secure Shell](#) and [Chapter 34, Secure Sockets Layer \(SSL\)](#).

Data Encryption

Data encryption for switches is driven by the need for organisations to keep sensitive data private and secure. Encrypting network data before it is passed to the wide area network (WAN) ensures that the data can not be read or modified as it crosses the WAN. Since all wide area traffic passes through the switch, the switch is the ideal place to locate the complex hardware required to provide secure data encryption. Locating the encryption function in the network switch and integrating the complex encryption key management procedures into the switch's management system minimises the cost of supporting an encrypted network.

Data encryption operates by applying an encryption algorithm and key to the original data (the plaintext) to convert it into an encrypted form (the ciphertext). The ciphertext produced by encryption is a function of the algorithm used and the key. Since it is easy to discover what type of algorithm is being used the security of an encryption system relies on the secrecy of its key information. When the ciphertext is received by the remote router the decryption algorithm and key are used to recover the original plaintext. Often a checksum is also added to the data before encryption to allow the validity of the data to be checked on decryption.

This section describes the following types of encryption:

- [Symmetrical Encryption](#)
- [Asymmetrical \(Public Key\) Encryption](#)
- [Payload Encryption](#)

Symmetrical Encryption

Symmetrical encryption refers to algorithms in which a single key is used for both the encryption and decryption processes. Anyone who has access to the key used to encrypt the plaintext can decrypt the ciphertext. Because the encryption key must be kept secret to protect the data these algorithms are also called private, or secret key algorithms. The key can be any value of the appropriate length.

DES The most common symmetrical encryption system is the *Data Encryption Standard* (DES) algorithm (FIPS PUB 46). The DES algorithm has proven to be a highly secure encryption algorithm. To fully conform to the DES standard the actual data encryption operations must be carried out in hardware. Software implementations can only be DES-compatible, not DES-compliant. The DES algorithm has a key length of 56 bits and operates on 64-bit blocks of data. DES can be used in the following modes:

Mode	Description
Electronic Code Book (ECB)	The fundamental DES function. Plaintext is divided into 64-bit blocks that are encrypted with the DES algorithm and key. ECB always produces the same block of ciphertext for a given input block of plaintext.
Cipher Block Chaining (CBC)	The most popular form of DES encryption. CBC operates on 64-bit blocks of data, but includes a feedback step that chains consecutive blocks so that repetitive plaintext data (such as ASCII blanks) does not yield identical ciphertext. CBC introduces a dependency between data blocks, which protects against fraudulent data insertion and replay attacks. Feedback for the first block of data is provided by a 64-bit Initialisation Vector (IV). This is the DES mode used for the switch's data encryption process.
Cipher FeedBack (CFB)	An additive stream cipher method that uses DES to generate a pseudo-random binary stream that is combined with the plaintext to produce the ciphertext. The ciphertext is then fed back to form a portion of the next DES input block.
Output FeedBack (OFB)	Combines the first IV with the plaintext to form ciphertext. The ciphertext is then used as the next IV.

The DES algorithm has been optimised to produce very high speed hardware implementations, making it ideal for networks where high throughput and low latency are essential.

Triple DES The Triple DES (3DES) encryption algorithm is a simple variant on the DES CBC algorithm. The DES function is replaced by three rounds of that function, an encryption followed by a decryption, which is followed by an encryption. This can be done by using either two DES keys (112-bit key) or three DES keys (168-bit key).

The two-key algorithm encrypts the data with the first key, decrypts it with the second key, and then encrypts the data again with the first key. The three-key algorithm uses a different key for each step. The three-key algorithm is the most secure algorithm due to the longer key length.

There are several modes in which Triple DES encryption can be performed. The most common modes are:

- **Inner CBC mode** encrypts the entire packet in CBC mode three times, and requires three different initialisation vectors (IV's).
- **Outer CBC mode** triple encrypts each 8-byte block of a packet in CBC mode three times and requires one IV.

A special feature licence is required to use Triple DES encryption. For details contact your authorised distributor or reseller.

Asymmetrical (Public Key) Encryption

Asymmetrical encryption algorithms use two keys—one for encryption and one for decryption. The encryption key is called the *public key* because it cannot be used to decrypt a message and therefore does not have to be kept secret. Only the decryption, or private key needs to be kept secret. The other name for this type of algorithm is *public key encryption*. The public and private key pair cannot be randomly assigned but must be generated together. In a typical scenario, a decryption station will generate a key pair and then distribute the public key to encrypting stations. This distribution need not be kept secret, but must be protected against the substitution of the public key by a malicious third party.

Another use for asymmetrical encryption is as a digital signature. The signature station publishes its public key, and then signs its messages by encrypting them with its private key. To verify the source of a message the receiver decrypts the messages with the published public key. If the message that results is valid then the signing station is authenticated as the source of the message.

The most common asymmetrical encryption algorithm is RSA. RSA utilises mathematical operations which are relatively easy to calculate in one direction but which have no known reverse solution. The security of RSA relies on the difficulty of factoring the modulus of the RSA key. Because typical key lengths of 512 bits or greater are used in public key encryption systems, decrypting RSA encrypted messages is almost impossible with current technology.

Asymmetrical encryption algorithms require enormous computational resources, making them very slow when compared to symmetrical algorithms. For this reason they are normally only used on small blocks of data (e.g. exchanging symmetrical algorithm keys), and not for entire data streams.

Payload Encryption

Payload encryption encrypts packet data at the network layer without changing the packet header. Since the routing information remains unchanged the packet can be carried across a routed network, such as an internet, without requiring intermediate switches to support the encryption algorithm or have any knowledge about how to access the secure data. This allows insecure internets to be used as a highly secure networks to transport sensitive information.

As well as preventing unauthorised viewing and modification of the data, payload encryption can also prevent unauthorised access into the network. Only a switch with knowledge of secret encryption keys can access the network. All other attempts will decrypt incorrectly and be discarded. Such functionality is very difficult for an external device to achieve because the device would need an understanding of the network layer protocol before it could tell which part of the packet to encrypt or decrypt.

Authentication

Data authentication for switches is driven by the need for organisations to verify that sensitive data has not been altered. Authenticating network data before it is passed to the wide area network (WAN) ensures that the data can not be altered as it traverses the WAN. Since all wide area traffic passes through the switch, the switch is the ideal place to locate the complex processing required to provide secure data authentication. Locating the authentication function in the network switch and integrating the complex authentication key management procedures into the switch's management system minimises the cost of supporting an authenticated network.

Data authentication operates by calculating a Message Authentication Code (MAC), commonly referred to as a *hash*, of the original data and appending it to the message. The MAC produced is a function of the algorithm used and the key. Since it is easy to discover what type of algorithm is being used the security of an authentication system relies on the secrecy of its key information. When the message is received by the remote switch another MAC is calculated and checked against the MAC appended to the message. When the two MACs are identical, the message is authentic.

Typically a MAC is calculated using a keyed one-way hash algorithm. A keyed one-way hash function operates on an arbitrary-length message and a key and returns a fixed length hash. The following properties make the hash function one-way:

- easy to calculate the hash from the message and the key
- very hard to compute the message and the key from the hash
- very hard to find another message and key that give the same hash

The two most commonly used one-way hash algorithms are:

- MD5 (Message Digest 5) defined in RFC 1321
- SHA-1 (Secure Hash Algorithm) defined in FIPS-180-1)

MD5 returns a 128-bit hash and SHA-1 returns a 160-bit hash. MD5 is faster in software than SHA-1, but SHA-1 is generally regarded to be slightly more secure.

HMAC is a mechanism for calculating a keyed Message Authentication Code which can use any one-way hash function. It allows for keys to be handled the same way for all hash functions and it allows for different sized hashes to be returned.

Another method of calculating a MAC is to use a symmetric block cypher such as DES in CBC mode. This is done by encrypting the message and using the last encrypted block as the MAC and appending this to the original message (plain-text). Using CBC mode ensures that the whole message affects the resulting MAC. See [“DES” on page 30-3](#) for more information about DES in CBC mode.

Key Exchange Algorithms

Key exchange algorithms are used by switches to securely generate and exchange encryption and authentication keys with other switches. Without key exchange algorithms, encryption and authentication session keys must be manually changed by the system administrator. If it is not possible to gain physical access to all switches in the secure network, it is virtually impossible to do this securely. Key exchange algorithms enable switches to re-generate session keys automatically and on a frequent basis.

The most important property of any key exchange algorithm is that only the negotiating parties are able to decode or generate the shared secret. Because of this requirement, public key cryptography plays an important role in key exchange algorithms. Public key cryptography provides a method of encrypting a message which can only be decrypted by one party. A switch can generate a session key, encrypt the key using public key cryptography, transmit the key over an insecure channel, and be certain that the key can only be decrypted by the intended recipient. Symmetrical encryption algorithms can also be used for key exchange, but commonly require an initial shared secret to be manually entered into all switches in the secure network.

The *Diffie-Hellman* algorithm is one of the more commonly used key exchange algorithms. It is not an encryption algorithm because messages can not be encrypted using Diffie-Hellman. Instead it provides a method for two parties to generate the same shared secret with the knowledge that no other party can generate that same value. It uses public key cryptography and is commonly known as the first public key algorithm. Its security is based on the difficulty of solving the “discrete logarithm problem”, which can be compared to the difficulty of factoring very large integers.

A Diffie-Hellman algorithm requires more processing overhead than RSA based key exchange schemes, but does not need the initial exchange of public keys. Instead, it uses published and well tested public key values. The security of the Diffie-Hellman algorithm depends on these values. Public key values less than 768 bits long are considered to be insecure.

A Diffie-Hellman exchange starts with both parties each generating a large random number. These values are kept secret, while the result of a public key operation on the random number is transmitted to the other party. A second public key operation, this time using the random number and the exchanged value, results in the shared secret. As long as no other party knows either of the random values, the secret is safe.

ENCO Services

The ENCO module provides services to user applications via channel pairs. A user application requests a service, specifying any configuration needed for the service, and is attached to an ENCO channel pair if the service and free channels are available. A channel pair consists of an encoding channel and a decoding channel. An encoding channel is used for encryption, and a decoding channel is used for decoding.

The ENCO module provides the following services:

- Encryption—DES encryption, 112-bit 2-key Outer mode Triple DES encryption, 168-bit Inner mode Triple DES encryption, RSA encryption, and RSA key generation
- Authentication—HMAC MD5-96, HMAC SHA-1-96, and DES-MAC
- Diffie-Hellman key exchange
- Key storage

The number of channels available depends on the amount of RAM on the switch. Switches with up to 8 MBytes of RAM can have up to 512 encryption channels. Switches with 16 MBytes can have up to 1024 channels, and switches with 32 Mbytes up to 2048 channels. To check the amount of RAM on a switch, use the [show system command on page 4-54 of Chapter 4, Configuring and Monitoring the System](#).

The identification number of the lowest and highest channels available can be displayed by using the [show enco command on page 30-18](#). This command also displays general information about the ENCO module and available services.

A user module that requests the retention of process histories between packets for an encryption service may also request that the history of one of its channels be reset. Whenever a decoding channel gets out of step with its associated encoding channel the encoding channel's history must be reset.

Encryption

The ENCO module provides the following variants of DES encryption:

- 56-bit single DES
- 112-bit 2-key Outer CBC Mode Triple DES
- 168-bit Inner CBC Mode Triple DES
- 168-bit Outer CBC Mode Triple DES

For management purposes, the ENCO module provides RSA and DES encryption to the Secure Shell and Secure Sockets Layer modules. SSL uses RSA indirectly because it uses PKI, and PKI uses RSA. For more information, see [Chapter 33, Secure Shell](#), [Chapter 34, Secure Sockets Layer \(SSL\)](#), and [Chapter 35, Public Key Infrastructure \(PKI\)](#).

Authentication

The ENCO module provides three authentication algorithms—HMAC MD5-96, HMAC SHA-1-96, and DES-MAC. The HMAC hash algorithms are provided in software. The DES-MAC hash algorithm is provided in hardware and requires the presence of an encryption-capable PAC.

Diffie-Hellman Key Exchange Algorithm

The ENCO module provides the Diffie-Hellman key exchange algorithm in software. The exchange takes place in the following phases:

1. A local random number is generated, combined with a Diffie-Hellman public key value and transmitted to the other party in the key exchange.
2. A shared secret is generated from the exchanged and local random number values.

Each phase is handled as smaller bits of processing because of the processor-intensive public key calculations. As a result the key exchange takes longer but the general operation of the switch continues during the process.

The Diffie-Hellman key exchange has a high priority by default. If the speed of the key exchange is not critical, the priority can be set to a lower value to leave more CPU time available for routing processes. To change the Diffie-Hellman key exchange priority, use the [set enco dhpriority command on page 30-16](#).

The ENCO module supports published public key values “MODP Group 1” and “MODP Group 2” as defined in RFC 2412, *The OAKLEY Key Determination Protocol*. These public key values are 768 and 1024 bits long, respectively.

Key Creation and Storage

Keys required by the switch for encryption and authentication services are created and stored by the ENCO module. Keys are stored in the flash memory.

The following types of keys are used by the switch:

- DES, used for DES encryption
- 3DES2KEY, used for 112-bit 2-key Outer CBC Mode Triple DES encryption
- 3DESINNER, used for 168-bit Inner CBC Mode Triple DES encryption
- RSA, used for RSA encryption
- GENERAL, used for HMAC authentication

Each key that is created and stored in the switch has a unique identification number. Each key has a set of attributes, including the user module associated with the key. Each user module specifies whether these key attributes must be set. For example, SSH uses the IP address attribute of RSA keys to find the public key of the remote peer.

Key Formats

Keys can be stored in the switch and displayed in several formats.

DES, 3DES2KEY, and 3DESINNER keys can be entered in a proprietary short/checked ASCII format or in hexadecimal format. The short/checked ASCII format represents each 5 bits of the key by an ASCII character. The legal characters are lowercase letters and digits (2–9). The digits 0 and 1 are not used,

to prevent confusion with the letters O and I. A checksum field is added to ensure the key has been entered correctly. DES, 3DES2KEY, and 3DESINNER keys are displayed in both hexadecimal and the short/checked ASCII format.

RSA keys can not be entered via the command line. They can be generated on the switch or imported from a text file. See [“RSA Keys” on page 30-9](#) and the [create enco key command on page 30-11](#) for more information about importing RSA keys.

General keys can be entered in hexadecimal format or as a pass phrase that is easy to remember. The phrase must contain only printable characters and must be in double quotes if it contains spaces. General keys are displayed in hexadecimal format and also in string format if the key data contains all printable characters.

See the [create enco key command on page 30-11](#) for more information about generating and entering keys into the switch. See the [show enco key command on page 30-35](#) for more information about displaying keys.

RSA Keys

RSA keys can be generated using the [create enco key command on page 30-11](#). RSA public keys generated on an external device (such as a PC) can be imported as a text file. RSA public keys on the switch can be exported to a text file for transfer to an external device.

Generating RSA keys is time consuming and processor intensive. Two very large random prime numbers must be created and then combined to form the RSA key. The fastest method for generating large prime numbers is to create a large random number and then test to see if it is prime. If is not, then the number is modified and tested again. Because this is a very random procedure, it can take anywhere between 3 seconds and 30 minutes, depending on the size of the key and the CPU. To ensure the normal operation of the switch is not effected, RSA key generation is performed as a background task.

RSA public keys can be imported from and exported to text files. RSA public keys generated by an external device can be loaded on to the switch in text file format (.key files) and ENCO RSA public keys are generated from these text files. When an RSA key generated by the switch is exported to a text file, only the public portion of the key is written to the file.

The switch recognises three text file formats—SSH ([Figure 30-1](#)), NIQ ([Figure 30-2](#)) and hex ([Figure 30-3](#)). The default format is hex. The first number in the key file is the length of the RSA public key in bits; the second number is the exponent field of the key; the last number is the modulus field of the key. The length, exponent, and modulus fields in SSH format files are in one long line. In [Figure 30-1](#) the backward slash (**bold**) shows where the line wraps. The length, exponent, and modulus fields are on separate lines in NIQ and hex formats.

Figure 30-1: An RSA public key file in SSH format

```
512 65537 58271454040942172675574803018707732886250732940593381153466514993637269\
2554308139731130814782897798791374252039162251634873178364999125511405069275595877
```

Figure 30-2: An RSA public key file in NIQ format

```
-NiQ Switch RSA Public Key
512
65537
5827145404094217267557480301870773288625
0732940593381153466514993637206925543081
3973113081478289779879137425203916225163
4873178364999125511405069275595877
```

Figure 30-3: An RSA public key file in HEX format

```
512
0x010001
0x6f427e5112e1389e2af1c4df09545fa88f90b093aabbdebb5778ef5ed1d39fe9
248602ef11e399216b52adae2f5fd1ae8b7ca5c19b3c27a3ec5179966cb58465
```

See the [create enco key command on page 30-11](#) for more information about generating and entering keys into the switch.

Access Control

When encryption is configured and enabled, the switch must be in *security mode*. See the [enable system security_mode command on page 29-43 of Chapter 29, User Authentication](#) for details.



Caution Keys created on a switch not already in security mode are destroyed when the switch is restarted. Enable security mode before creating encryption keys.

When the switch is in security mode, only users with security officer privilege can execute commands that could impact the security of the switch and its keys.

A user must login from a local port, a Secure Shell session or a Telnet session from a remote security officer address (if the Remote Security Officer function is enabled) to gain security officer privilege.

See [Chapter 29, User Authentication](#) for more information about creating users with security officer privilege, configuring remote security officers, and logging into a user account with security officer privilege. See [Chapter 33, Secure Shell](#) for more information about Secure Shell.

When an encrypting switch is removed from service its sensitive encryption keys should be cleared before it is transported in an unprotected manner. To clear all encryption keys on a switch by disabling security mode, use the [disable system security_mode command on page 29-39 of Chapter 29, User Authentication](#).

Important The encryption technology used in the switch is a government controlled product. The switch encryption hardware must never be disposed of by the user. These products must always be returned to your authorised distributor or reseller for deregistration and disposal.

Command Reference

This section describes the commands available on the switch to configure and monitor the encryption processes on the switch.

A user must be logged in with security officer privilege to configure encryption services. See [Chapter 29, User Authentication](#) for more information about creating users with security officer privilege, configuring remote security officers, and logging in with security officer privilege.

The shortest valid command is denoted by capital letters in the Syntax section. See [“Conventions” on page xlix of About this Software Reference](#) in the front of this manual for details of the conventions used to describe command syntax. See [Appendix A, Messages](#) for a complete list of messages and their meanings.

create enco key

Syntax CREate ENCo KEY=*key-id* TYPE={DES|3DES2key|3DESInner|GENERAL|RSA} [DESCRiption=*description*] [FILE=*filename*] [FORMat={HEX|NIQ|SSH}] [IPaddress=*ipadd*] [LENGth=*length*] [MODule=*module-id*] [{RANDom|VALUE=*value*}]

where:

- *key-id* is a number from 0 to 65535.
- *description* is a character string 1 to 24 characters long. Valid characters are any printable character. If *description* contains spaces, it must be in double quotes.
- *filename* is a valid switch filename with a .key extension.
- *ipadd* is an IP address in dotted decimal notation.
- *length* is a number from 0 to 65535.
- *module-id* is the name or number of a switch module (see [“Module Identifiers and Names” on page B-2 of Appendix B, Reference Tables](#) for a complete list).
- *value* is a character string of variable length depending on the key type. For ASCII formatted keys, valid characters are lowercase letters and digits (2-9). The digits 0 and 1 are illegal, to prevent confusion with the letters O and I. Hexadecimal keys must start with “0x” and must contain an even number of characters using the digits (0-9), and letters (a-f). Passphrase keys may contain any printable character. If *value* contains spaces, it must be in double quotes.

Description This command creates a specific type of encryption key, and stores the key information in the switch’s flash memory. You can also import or export RSA keys with this command. This command requires a user with security officer privilege when the switch is in security mode.

The **key** parameter specifies the identification number for the key.

The **type** parameter specifies the type of key to be created. If a DES or 3DES key is being created, then the **random** or **value** parameters must be specified. If **des** is specified, a 56-bit DES key is created. If **3des2key** is specified, a 112-bit DES

key is created. If **3desinner** is specified, a 168-bit DES key is created. If an RSA key is being generated, then the **length** or **file** parameters must be specified. If the **file** parameter is specified, the RSA key is imported from or exported to the specified file. If the **file** parameter is not specified, then a random RSA key is generated. If a **general** key is being created, then the **length** or **value** parameters must be specified. Keys of the **general** type can be used for authentication algorithms or as shared secrets.

The **description** parameter specifies a user-defined description for the key, to make it easier to keep track of different keys.

The **file** parameter specifies name of a switch file. RSA public keys may be imported from or exported to a file in either Secure Shell format, the switch's own format or in hexadecimal format. If the file exists but the specified RSA key does not exist, then the RSA key is imported from the file. If the specified RSA key does exist but the file does not exist, the RSA key is exported to the file. The **format** parameter must be specified when importing or exporting keys.

The **format** parameter specifies the format of the key file when importing or exporting an RSA key. Secure Shell users should use SSH. NIQ is the switch's own format, which can be used for transferring RSA keys between switches. Hex format should be used when transferring keys between other devices. The default is **hex**. If **format** is specified, the **file** parameter must also be present.

The **ipaddress** parameter specifies an IP address to associate with the key. The ISAKMP and SSH modules use this value to find the RSA public key of a remote peer. Documentation for particular modules specify whether this parameter is required.

The **length** parameter specifies the length of the key to be created. An RSA key length is specified in bits and must be a multiple of 32. Valid RSA keys are from 256 to 2048 bits. A key of **general** type can have any length from 2 to 64 bytes.

The **module** parameter can be used to link a key to a specific module. Documentation for particular modules specify whether this parameter is required.

The **random** parameter creates a random key. The key can entered into another switch by using the **create enco key** command and specifying the **value** parameter.

The **value** parameter creates a key with the supplied value. DES and 3DES keys require a key in 5-bit ASCII format or hexadecimal format. This ASCII representation includes a check value of the key to ensure it has been typed in correctly. A hexadecimal key always starts with "0x". The value of a key can be displayed by using the **show enco key** command. Keys of **general** type can be entered as a string or in hexadecimal format.

Examples To create a random DES encryption key with the identification number 1 and then display the key, use the commands:

```
cre enc key=1 typ=des rand
sh enc key=1
```

To add this key to the other switch, use the following command on the other switch:

```
cre enc key=1 type=des value=value
```

where *value* is the value of the key displayed in the output of the **show enco key** command.

To create a random 512-bit RSA private key with the key identification number 2, use the command:

```
cre enc key=2 typ=rsa len=512 desc="Switch A private key"
```

To create an uploadable file for the public component of the same RSA key in the format used by Secure Shell use the command:

```
cre enc key=2 typ=rsa fil=routerA.key for=ssh
```

To import an RSA key from the file `rsa.key`, which is in hex format, as encryption key 3, use the command:

```
cre enc key=3 typ=rsa fil=rsa.key for=hex
```

Related Commands [destroy enco key](#)
[set enco key](#)
[show enco key](#)

destroy enco key

Syntax DESTroy ENCo KEY=*key-id* LOCation=FLAsh

where *key-id* is a number from 0 to 65535

Description This command destroys the specified encryption key. The memory the key occupied is overwritten to ensure the key is irretrievable. This command requires a user with security officer privilege when the switch is in security mode.

The **key** parameter specifies the identification number for the key. A key with the specified identification number must exist.

The **location** parameter specifies the location of the key.

Examples To destroy the encryption key in flash with the key identification number 4, use the command:

```
dest enc key=4 loc=fla
```

Related Commands [create enco key](#)
[set enco key](#)
[show enco key](#)

disable enco compstatistics

Syntax DISable ENCo COMPSTatistics

Description This command disables the calculation and storage of compression ratio statistics for any compression-only ENCO channels.

Related Commands [enable enco compstatistics](#)
[show enco channel](#)

disable enco debugging

Syntax DISable ENCo DEBUgging={CHannel | PAcKet | TImeStamp | ALL}

Description This command disables debugging for the ENCO module.

The **debug** parameter specifies the debugging option to disable. Specify **packet** to disable debugging of the contents of packets processed by the ENCO module. Specify **timestamp** to disable measurements of encryption and compression operations. Specify **all** to disable all debugging options. The **channel** option is no longer supported.

Examples To disable the debugging of the contents of packets processed by the ENCO module, use the command:

```
dis enc deb=pa
```

Related Commands [enable enco debugging](#)
[show enco debug](#)

enable enco compstatistics

Syntax ENAbLe ENCo COMPSTatistics

Description This command enables the calculation and storage of compression ratio statistics for any compression-only ENCO channels.

Related Commands [disable enco compstatistics](#)
[show enco channel](#)

enable enco debugging

Syntax ENABle ENCo DEBugging={CHannel | PAcKet | TImeStamp | ALL}

Description This command enables debugging for the ENCO module. Debugging information is sent to the terminal from which the command was entered.

The **debug** parameter specifies the debugging option to enable. Specify **packet** to enable debugging of the contents of packets processed by the ENCO module. Specify **timestamp** to enable measurements of encryption and compression operations. Specify **all** to enable all debugging options. The **channel** option is no longer supported.

Examples To enable the debugging of the contents of packets processed by the ENCO module, use the command:

```
ena enc deb=pa
```

Related Commands [disable enco debugging](#)
[show enco debug](#)

reset enco counter

Syntax RESET ENCo COUNTER={ALl | DEs | DH | HMac | JObprocessing | QUeues | RSa | SSl | USeR | UTil}

Description This command clears general and process-specific counters for the ENCO module. It requires a user with security officer privilege when the switch is in security mode.

The **counter** parameter specifies the category of counters to be cleared. If a category is not specified, all ENCO counters are cleared. The **user**, **util**, **queue** and **jobprocessing** counters display information about the general operation of the ENCO module. The other counters display information for particular processes.

If **all** is specified, all counters are reset. If **des** is specified, counters for the DES encryption process are reset. If **dh** is specified, counters for the Diffie-Hellman key exchange process are reset. If **hmac** is specified, counters for the HMAC message process are reset. If **jobprocessing** or **queues** is specified, counters are reset for the jobs that have been or are still being processed by the ENCO module. If **rsa** is specified, counters are reset for the RSA encryption process. If **ssl** is specified, counters for the SSL process are reset. If **user** is specified, counters are reset for the interface between ENCO and user applications that use ENCO channels. If **util** is specified, counters are reset for the interface between the ENCO module and other user applications that use the ENCO module for one-off jobs.

Examples To reset all the ENCO counters, use the command:

```
reset enc cou
```

To reset all the counters for the DES encryption process, use the command:

```
reset enc cou=de
```

Related Commands [show enco counter](#)

set enco dhpriority

Syntax SET ENCo DHPriority={High|Medium|Low}

Description This command changes the priority of the Diffie-Hellman key exchange on the switch. The higher the priority, the more CPU time the algorithm can use and the sooner the key exchange finishes. If the speed of the key exchange is not critical, **dhpriority** can be set lower so that the routing process receives more CPU time.

The **dhpriority** parameter specifies the priority of the Diffie-Hellman key exchange. The default priority is **high**.

Examples To set the Diffie-Hellman priority to low, use the command:

```
set enc dhpr=1
```

Related Commands [show enco key](#)

set enco key

Syntax SET ENCo KEY=*key-id* [DESCRiption=*description*]
[IPaddress=*ipadd*] [MODule=*module-id*]

where:

- *key-id* is a number from 0 to 65535.
- *description* is a character string 1 to 24 characters long. Valid characters are any printable character. If *description* contains spaces, it must be in double quotes.
- *ipadd* is an IP address in dotted decimal notation.
- *module-id* is the name or number of a switch module (see “[Module Identifiers and Names](#)” on page B-2 of [Appendix B, Reference Tables](#) for a complete list).

Description This command changes the user-defined description, IP address, or module for a specific key. It requires a user with security officer privilege when the switch is in security mode.

The **key** parameter specifies the identification number for the key. The specified encryption key must already exist.

The **description** parameter specifies a user-defined description for the key, to make it easier to keep track of different keys.

The **ipaddress** parameter specifies an IP address to associate with the key. The SSH module uses this value to find the RSA public key of a remote peer. Documentation for individual modules specify whether this parameter is required.

The **module** parameter can be used to link a key to a specific module. Documentation for particular modules specify whether this parameter is required.

Examples To change the description for key 1, use the command:

```
set enc key=1 descr="Switch Z key"
```

Related Commands [create enco key](#)
[destroy enco key](#)
[show enco key](#)

show enco

Syntax SHow ENCo

Description This command displays information about the ENCO module ([Figure 30-4](#), [Table 30-1](#)).

Figure 30-4: Example output from the **show enco** command

```
ENCO Module Configuration

Hardware ..... NOT PRESENT
Lowest valid channel ..... 1
Highest valid channel ..... 511
Compression Statistics Enabled ..... FALSE
```

Table 30-1: Parameters in output of the **show enco** command

Parameter	Meaning
Hardware	Encryption hardware, if available.
Lowest valid channel	Identification number of the lowest channel available for use by a user module.
Highest valid channel	Identification number of the highest channel available for use by a user module.
Compression Statistics Enabled	Whether gathering of compression statistics is enabled for all channels.
Diffie Hellman Priority	Whether the priority of the Diffie-Hellman key exchange is high, medium, or low.

Related Commands [set enco dhpriority](#)
[show enco channel](#)
[show enco counter](#)

show enco channel

Syntax `SHoW ENCo CHannel [=channel [COUnTer]]`

where channel is a number from 0 to 512, if the switch has up to 8Mbytes of RAM, 0 to 1024 if the switch has 16Mbytes of RAM, or 0 to 2048 if the switch has 32Mbytes of RAM.

Description This command displays information about active ENCO module channels. If no ENCO channel is specified, a summary of all currently active channels is displayed (Figure 30-5, Table 30-2). If a channel is specified, detailed configuration and status information is displayed about it (Figure 30-6 on page 30-20, Table 30-3 on page 30-20).

If the **counter** parameter is specified, information counters are displayed for the specified channel (Figure 30-7 on page 30-22, Table 30-4 on page 30-22).

Figure 30-5: Example output from the **show enco channel** command

Channel	State	User	UserID	MDL	pktOverhead	Process
1	UP	SA	f0000001	1528	72	DES
1	UP	SSH	00000001	1584	16	DES

Table 30-2: Parameters in output of the **show enco channel** command

Parameter	Meaning
Channel	Channel identification number.
State	Whether the channel is up or down.
User	Whether the user module attached to this channel is TEST or SSH.
UserID	Number used by the user module to identify this channel.
MDL	Maximum data length of packets accepted on this channel.
pktOverhead	Number of bytes that the user module requested be reserved in a packet in front of encoded data.
Process	The process for which the channel is configured: RSA DH DES HMAC

Figure 30-6: Example output from the **show enco channel** command for a specific channel.

```

Channel ..... 1

Type ..... ENCODE/DECODE
State ..... UP
User ..... SSH
User ID ..... 00000001
Maximum Data Length ..... 1584
Packet Overhead ..... 16
Process ..... DES
Process Configuration:
  Des Type.....DES - 56 bit
  Check Type .....NONE
  Channel Type.....ENCODE/DECODE
  History Mode.....Off
  IV Type.....Random

```

Table 30-3: Parameters in output of the **show enco channel** command for a specific channel

Parameter	Meaning
Channel	Identification number of the channel.
Type	Mode of the channel: ENCODE/DECODE ENCODE ONLY DECODE ONLY
State	Whether the channel is up or down.
User	Whether the user module attached to this channel is TEST or SSH.
User ID	A number used by the user module to identify this channel.
Maximum Data Length	The maximum data length of packets accepted on this channel.
Packet Overhead	The number of bytes reserved at the head of data packets in front of the encoded data, for lower layer packet headers.
Process	The process for which the channel is configured: RSA DH DES
Process Configuration	Details about a particular process. The fields displayed vary depending on the process.
Max Data Length	The maximum allowed length of data packets on the channel.
Check Type	The type of checksum to be used: XOR8 NONE CRCCITT

Table 30-3: Parameters in output of the **show enco channel** command for a specific channel (Continued)

Parameter	Meaning
DES Type	[DES] The DES encryption/decryption algorithm used to process packets on the channel: DES-56 bit 3DES-112 bit-outer CBC 3DES-168 bit-inner CBC 3DES-168 bit-outer CBC.
Channel Type	[DES] The mode of the channel: ENCODE/DECODE ENCODE ONLY DECODE ONLY.
History Mode	[DES] Whether DES is operating with history mode enabled.
IV Type	[DES] Whether the type of Initialisation Vector (IV) DES uses is zero, random, or specified.
RSA mode	[RSA] Whether the RSA encryption mode on this channel is public or private.
Mode	[Diffie-Hellman] Whether the mode is Phase 1 or Phase 2.
Group Type	[Diffie-Hellman] The group types supported. MODP is currently supported.
Group	[Diffie-Hellman] The Diffie-Hellman group: 768-bit MODP 1024-bit MODP
Algorithm	[HMAC] Whether the HMAC algorithm is MD5 or SHA.
Key Length	[HMAC] The length of the HMAC key.
Compression Statistics	Statistics for the compression process. This section is only displayed when compression statistics have been enabled with the enable enco compstatistics command.
Number of Packets Compressed	Number of data packets that have been compressed.
Best Compression Ratio	Highest compression ratio achieved.
Mean Compression Ratio	Mean compression ratio achieved.
Worst Compression Ratio	Lowest compression ratio achieved.
Compression Ratio	Range of compression ratios.
Number of Packets	Number of packets compressed, for which the resulting compression ration was in the specified range.

Figure 30-7: Example output from the **show enco channel counter** command

Channel Counter:			
UP events	1	DOWN events	0
start config	1	attach good	1
encode NULL packets	0	decode NULL packets	0
encode bad priorities	0	decode bad priorities	0
encode bad length	0	decode bad length	0
encode actions sent	0	decode actions sent	0
good encodes	0	good decodes	0
bad encodes	0	bad decodes	0
reset E actions sent	0	reset D actions sent	0
good encode resets	0	good decode resets	0
bad encode resets	0	bad decode resets	0
discarded encode jobs	0	discarded decode jobs	0

Table 30-4: Parameters in output of the **show enco channel counter** command

Parameter	Meaning
UP events	Number of times the channel has entered an up state.
DOWN events	Number of times the channel has entered a down state.
start config	Number of times a configure operation has started on the channel.
attach good	Number of successful attach operations on the channel.
encode NULL packets	Number of encode requests received from a user module with no data packet.
decode NULL packets	Number of decode requests received from a user module with no data packet.
encode bad priorities	Number of encode requests received from a user module with a data packet containing an unknown priority.
decode bad priorities	Number of decode requests received from a user module with a data packet containing an unknown priority.
encode bad length	Number of encode requests received from a user module with a data packet with a bad length.
decode bad length	Number of decode requests received from a user module with a data packet with a bad length.
encode actions sent	Number of encode actions which have been sent to the process on this channel.
decode actions sent	Number of decode actions which have been sent to the process on this channel.
good encodes	Number of successful encode operations on the channel.
good decodes	Number of successful decode operations on the channel.
bad encodes	Number of unsuccessful encode operations on the channel.
bad decodes	Number of unsuccessful decode operations on the channel.
reset E actions sent	Number of encode reset actions which have been sent to the process on the channel.
reset D actions sent	Number of decode reset actions which have been sent to the process on the channel.
good encode resets	Number of successful encode resets on the channel.

Table 30-4: Parameters in output of the **show enco channel counter** command (Continued)

Parameter	Meaning
good decode resets	Number of successful decode resets on the channel.
bad encode resets	Number of unsuccessful encode resets on the channel.
bad decode resets	Number of unsuccessful decode resets on the channel.
discarded encode jobs	Number of encode jobs discarded due to queue overloading or a channel reset.
discarded decode jobs	Number of decode jobs discarded due to queue overloading or a channel reset.

Examples To show a summary of all active ENCO channels, use the command:

```
sh enc ch
```

To show detailed configuration and status information for channel 1, use the command:

```
sh enc ch=1
```

To show counter information for channel 1, use the command:

```
sh enc ch=1 cou
```

Related Commands [set enco dhpriority](#)
[show enco](#)
[show enco counter](#)

show enco counter

Syntax SHow ENCo CUNter={ALl | DEs | DH | HMac | JObprocessing | QUeues | RSa | SSl | USer | UTil}

Description This command displays information counters for the ENCO module.

The **counter** parameter specifies the category of counters to display. The **user**, **util**, **queues** and **jobprocessing** options display information about the general operation of the ENCO module. The other options display information for particular processes.

If **all** is specified, all counters are displayed.

If **des** is specified, counters for the DES encryption process are displayed ([Figure 30-8 on page 30-25](#), [Table 30-5 on page 30-25](#)).

If **dh** is specified, counters for the Diffie-Hellman key exchange process are displayed ([Figure 30-9 on page 30-27](#), [Table 30-6 on page 30-27](#)).

If **hmac** is specified, counters for the HMAC message process are displayed ([Figure 30-10 on page 30-27](#), [Table 30-7 on page 30-27](#)).

If **jobprocessing** or **queues** is specified, counters are displayed for the jobs that have been or are still being processed by the ENCO module ([Figure 30-11 on page 30-28](#), [Table 30-8 on page 30-29](#)).

If **rsa** is specified, counters are displayed for the RSA encryption process ([Figure 30-12 on page 30-29](#), [Table 30-9 on page 30-29](#)).

If **ssl** is specified, counters for the SSL process are displayed ([Figure 30-13 on page 30-30](#), [Table 30-10 on page 30-30](#)).

If **user** is specified, counters are displayed for the interface between ENCO and user applications that use ENCO channels ([Figure 30-14 on page 30-32](#), [Table 30-11 on page 30-33](#)).

If **util** is specified, counters are displayed for the interface between ENCO and user applications that use the ENCO module for one-off jobs ([Figure 30-15 on page 30-34](#), [Table 30-12 on page 30-34](#)).

Figure 30-8: Example output from the **show enco counter=des** command

ENCO Process DES/3DES Counter:			
configGood	1	configBad	0
configNoResource	0	configNotSSH	0
BadBuffer	0	BadAlign	0
BadLength	0	nohistory	0
desJobs	0	3Des2KeyJobs	0
3DesInnerJobs	0	noHistJobs	0
desMacJobs	0		
badDesType	0	badJobType	0
unknownJob	0	error	0
reset	0	confNotDes	0
commWaitTimeOut	0	dataInWaitTimeOut	0
dataOutWaitTimeOut	0		
goodDecrypt	0	goodEncrypt	0
badDecrypt	0	badEncrypt	0
DMA1Start	0	DMA2Start	0
DMA1Done	0	DMA2Done	0
DMABed	0	DMABes	0
DMABrkp	0	DMAConf	0
DMA1TimeOut	0	DMA2TimeOut	

Table 30-5: Parameters in output of the **show enco counter=des** command

Parameter	Meaning
configGood	Number of successful channel configurations.
configBad	Number of unsuccessful configuration attempts.
configNoResource	Number of configure attempts without resources.
configNotSSH	Number of attempts to configure a software DES channel when the user was not Secure Shell.
badBuffer	Number of jobs received by the DES/3DES encryption algorithm unit with a bad buffer.
badAlign	Number of jobs received by the DES/3DES encryption algorithm unit with a bad alignment of the packet.
badLength	Number of jobs received by the DES/3DES encryption algorithm unit with a bad length (not a multiple of the DES block length).
nohistory	Number of jobs received by the DES/3DES encryption algorithm unit without valid history (IV's).
desJobs	Number of 56-bit DES jobs received by the DES/3DES algorithm unit.
3Des2KeyJobs	Number of 112-bit 3DES jobs received by the DES/3DES algorithm unit.
3DesInnerJobs	Number of 168-bit 3DES jobs received by the DES/3DES algorithm unit.
noHistJobs	Number of jobs processed by the DES/3DES encryption algorithm unit with history mode set to OFF.
desMacJobs	Number of DES-MAC authentication jobs received by the DES/3DES algorithm unit.

Table 30-5: Parameters in output of the **show enco counter=des** command (Continued)

Parameter	Meaning
badDesType	Number of jobs received by the encryption algorithm unit with a invalid DES type.
badJobType	Number of jobs received by the DES/3DES encryption algorithm unit with an invalid job type.
unknownJob	Number of unknown jobs received by the DES/3DES encryption algorithm unit.
error	Number of errors that occurred in the DES/3DES encryption algorithm unit while processing data.
reset	Number of resets by the hardware encryption unit.
confNotDes	Number of attempts to configure a DES channel with an invalid encryption type.
commWaitTimeOut	Number of commands entered for the hardware encryption unit before it was ready for the new command.
dataInWaitTimeOut	Number of times the data was entered to the hardware encryption unit before it is ready for new data.
dataOutWaitTimeOut	Number of times data was read from the hardware encryption unit before it was ready to output new data.
goodDecrypt	Number of good decryption jobs processed by the DES/3DES algorithm unit.
goodEncrypt	Number of good encryption jobs processed by the DES/3DES algorithm unit.
badDecrypt	Number of bad decryption jobs processed by the DES/3DES algorithm unit.
badEncrypt	Number of bad encryption jobs processed by the DES/3DES algorithm unit.
DMA1Start	Number of times the DMA1 channel started.
DMA2Start	Number of times the DMA2 channel started.
DMA1Done	Number of times the DMA1 channel completed a transfer.
DMA2Done	Number of times the DMA2 channel completed a transfer.
DMABed	Number of times the Bus Error Destination occurred during DMA transfers.
DMABes	Number of times a Bus Error Source occurred during DMA transfers.
DMAbrkp	Number of times a DMA break point interrupt occurred.
DMAConf	Number of times a DMA configuration error occurred.
DMA1TimeOut	Number of times the DMA1 channel timeout occurred.
DMA2TimeOut	Number of times the DMA2 channel timeout occurred.

Figure 30-9: Example output from the **show enco counter=dh** command

ENCO Process Diffie-Hellman Counter:			
goodPhase1	1	badPhase1	0
goodPhase2	1	badPhase2	0
goodConfigure	2	badConfigure	0
badGroupType	0	badGroup	0
badGroupParameters	0	badDataLength	0
noResources	0	unknownJob	0

Table 30-6: Parameters in output of the **show enco counter=dh** command

Parameter	Meaning
goodPhase1	Number of good jobs for phase 1 of the D-H exchange.
badPhase1	Number of failed jobs for phase 1 of the D-H exchange.
goodPhase2	Number of good jobs for phase 2 of the D-H exchange.
badPhase2	Number of failed jobs for phase 2 of the D-H exchange.
goodConfigure	Number of good channel configurations.
badConfigure	Number of failed channel configurations.
badGroupType	Number of jobs with an invalid Group Type.
badGroup	Number of jobs with an invalid Group.
badGroupParameters	Number of jobs with invalid group parameters.
badDataLength	Number of jobs with a bad data length.
noResources	Number of configure jobs with no resources.
unknownJob	Number of unknown jobs.

Figure 30-10: Example output from the **show enco counter=hmac** command

ENCO Process MD5 Counter:			
goodHashMD5	1	badHashMD5	0
goodHashSHA	0	badHashSHA	0
goodConfigure	1	badConfigure	0
badAlgorithm	0	noResources	0
badKeyLength	0	unknownJob	0
badDataLength	0		

Table 30-7: Parameters in output of the **show enco counter=hmac** command

Parameter	Meaning
goodHashMD5	Number of good MD5 hashes.
badHashMd5	Number of failed MD5 hashes.
goodHashSHA	Number of good SHA hashes.
badHashSHA	Number of failed SHA hashes.
goodConfigure	Number of good channel configurations.
badConfigure	Number of failed channel configurations.
badAlgorithm	Number of channel configurations with invalid algorithm types.
noResources	Number of channel configurations with no resources

Table 30-7: Parameters in output of the **show enco counter=hmac** command (Continued)

Parameter	Meaning
badKeyLength	Number of jobs with an invalid key length.
unknownJob	Number of unknown jobs.
badDataLength	Number of jobs with an invalid data length.

Figure 30-11: Example output from the **show enco counter=jobprocessing** command.

Input queue lengths	
Immediate queue.....	0
Priority queue 0 (high).....	0
Priority queue 1.....	0
Priority queue 2.....	0
Priority queue 3.....	0
Priority queue 4.....	0
Priority queue 5.....	0
Priority queue 6.....	0
Priority queue 7.....	0
Priority queue 8 (low).....	0
Total actions queued.....	0
Lowest Input Priority Queue.....	4
Highest Input Priority Queue....	0
Input Queue Length Limit.....	30
Input Queue discards	
Immediate queue.....	0
Priority queue 0 (high).....	0
Priority queue 1.....	0
Priority queue 2.....	0
Priority queue 3.....	0
Priority queue 4.....	0
Priority queue 5.....	0
Priority queue 6.....	0
Priority queue 7.....	0
Priority queue 8 (low).....	0
Total Input Queue discards.....	0
Input Queue jobs processed	
Immediate queue.....	4
Priority queue 0 (high).....	0
Priority queue 1.....	0
Priority queue 2.....	0
Priority queue 3.....	0
Priority queue 4.....	310
Priority queue 5.....	0
Priority queue 6.....	0
Priority queue 7.....	0
Priority queue 8 (low).....	0
Total Input Queue jobs processed	314
Output queue length.....	0
Output queue jobs completed.....	310
Output queue discards.....	0

Table 30-8: Parameters in output of the **show enco counter=jobprocessing** command

Parameter	Meaning
Input queue lengths	Lengths of the ENCO module input queues.
Immediate queue	Number of actions on the immediate input queue.
Priority queue n	Number of actions on the specified priority input queue.
Total actions queued	Total number of actions on the input queues.
Lowest Input Priority Queue	Lowest input priority queue with queued actions.
Highest Input Priority Queue	Highest input priority queue with queued actions.
Input Queue Length Limit	Maximum length of the input queues.
Input Queue discards	Numbers of actions discarded from the input queues.
Immediate queue	Number of actions discarded from the immediate input queue.
Priority queue n	Number of actions discarded from the specified priority input queue.
Total Input Queue discards	Total number of actions discarded from the input queues.
Input Queue jobs processes	Numbers of jobs processed from the input queues.
Immediate queue	Number of jobs processed from the immediate input queue.
Priority queue n	Number of jobs processed from the specified priority input queue.
Total Input Queue jobs processed	Total number of jobs processed from the input queues.
Output queue length	Length of the output queue.
Output queue jobs completed	Number of jobs completed off the output queue.
Output queue discards	Number of jobs discarded from the output queue.

Figure 30-12: Example output from the **show enco counter=rsa** command

ENCO Process RSA Counter:			
goodPublicEncrypt	0	badPublicEncrypt	0
goodPrivateDecrypt	1	badPrivateDecrypt	0
goodPrivateEncrypt	0	badPrivateEncrypt	0
goodPublicDecrypt	0	badPublicDecrypt	0
goodGenerateKey	0	badGenerateKey	0
badDataLength	0	badKey	0

Table 30-9: Parameters in output of the **show enco counter=rsa** command

Parameter	Meaning
goodPublicEncrypt	Number of encryption jobs using an RSA public key.
goodPrivateDecrypt	Number of decryption jobs using an RSA private key.
goodPrivateEncrypt	Number of encryption jobs using an RSA private key.
goodPublicDecrypt	Number of decryption jobs using an RSA public key.
goodGenerateKey	Number of RSA keys that have been generated.
badDataLength	Number of jobs with a bad data length.
badPublicEncrypt	Number of failed encryption jobs using an RSA public key.

Table 30-9: Parameters in output of the **show enco counter=rsa** command (Continued)

Parameter	Meaning
badPrivateDecrypt	Number of failed decryption jobs using an RSA private key.
badPrivateEncrypt	Number of failed encryption jobs using an RSA private key.
badPublicDecrypt	Number of failed decryption jobs using an RSA public key.
badGenerateKey	Number of failed key generations.
badKey	Number of jobs where the RSA key was invalid.

Figure 30-13: Example output from the **show enco counters=ssl** command

```

ENCO Process SSL Counters:
  initialised ..... 1      initNoResources ..... 0
  configGood ..... 0      configNoConnection ..... 0
  configBadUserArgs ..... 0  configNoResources ..... 0
  destroyGood ..... 0      destroyNoConnection .... 0
  unknownJob ..... 0      doJobNoConnection ..... 0

Application Data:
  appDataEncoded ..... 0      appDataEncodeFail .... 0
  appDataHmacEncFail ... 0      appDataDesEncFail .... 0
  appDataDecoded ..... 0      appDataDecodeFail .... 0
  appDataDesDecFail .... 0      appDataHmacDecFail ... 0

Handshake:
  genMasterSecrtGood ... 0      genKeyMaterialGood ... 0
  ccsGood ..... 0      ccsFail ..... 0
  ccsDesConfigFail ..... 0      ccsHmacConfigFail .... 0
  processSKEGood ..... 0      processSKENoKey ..... 0
  procsSKECfgrSAFail ... 0      procsSKERSADecFail ... 0
  genCKEGood ..... 0      genCKENoKey ..... 0
  genCKEConfigRSAFail ... 0      genCKERSAEncFail ..... 0
  processCKEGood ..... 0      processCKENoKey ..... 0
  procsCKECfgrSAFail ... 0      procsCKERSADecFail ... 0
  genCVGood ..... 0      genCVNoKey ..... 0
  genCVConfigRSAFail ... 0      genCVRSAEncodeFail ... 0
  processCVGood ..... 0      processCVNoKey ..... 0
  procssCVCfgrSAFail ... 0      procssCVRSADecFail ... 0
  processCVFail ..... 0

```

Table 30-10: Parameters in the output of the **show enco counters=ssl** command

Parameter	Meaning
initialized	Number of Initialization operations performed.
initNoResources	Number of Initialization attempts without resources.
configGood	Number of successful channel configurations.
configNoConnection	Number of unsuccessful configuration attempts.
configBadUserArgs	Number of unsuccessful configuration attempts due to bad user arguments.
configNoResources	Number of unsuccessful configuration attempts due to lack of resources.
destroyGood	Number of successful channel configuration de-allocations.
destroyNoConnection	Number of channel configuration de-allocation attempts with no connection.

Table 30-10: Parameters in the output of the **show enco counters=ssl** command

Parameter	Meaning
unknownJob	Number of unknown jobs received.
doJobNoConnection	Number of jobs received with an invalid connection.
appDataEncoded	Number of successful attempts to encode application data.
appDataEncodeFail	Number of unsuccessful attempts to encode application data.
appDataHmacEncFail	Number of unsuccessful attempts to hash application data.
appDataDesEncFail	Number of unsuccessful attempts to encrypt application data.
appDataDecoded	Number of successful attempts to decode application data.
appDataDecodeFail	Number of unsuccessful attempts to decode application data.
appDataDesDecFail	Number of unsuccessful attempts to decrypt application data.
appDataHmacDecFail	Number of unsuccessful attempts to authenticate application data.
genMasterSecrtGood	Number of successful attempts to generate the Master Secret.
genKeyMaterialGood	Number of successful attempts to generate the key material.
ccsGood	Number of successful Change Cipher Spec's processed.
ccsFail	Number of unsuccessful Change Cipher Spec's.
ccsDesConfigFail	Number of unsuccessful Change Cipher Spec's due to failed DES configuration.
ccsHmacConfigFail	Number of unsuccessful Change Cipher Spec's due to failed HMAC configuration.
processSKEGood	Number of successful Server Key Exchange messages processed.
processSKENoKey	Number of failed SKE messages due to no key.
procsSKECfgRSAFail	Number of failed SKE messages due to failed RSA configuration.
procsSKERSADecFail	Number of failed SKE messages due to failed RSA decryption.
genCKEGood	Number of successful Client Key Exchange messages generated.
genCKENoKey	Number of unsuccessful CKE message generations due to no key.
genCKEConfigRSAFail	Number of unsuccessful CKE message generations due to failed RSA configuration.
genCKERSAEncFail	Number of unsuccessful CKE message generations due to failed RSA encryption.
processCKEGood	Number of successful Client Key Exchange messages processed.
processCKENoKey	Number of failed CKE messages due to no key.
procsCKECfgRSAFail	Number of failed CKE messages due to failed RSA configuration.

Table 30-10: Parameters in the output of the **show enco counters=ssl** command

Parameter	Meaning
procsCKERSADecFail	Number of failed CKE messages due to failed RSA decryption.
genCVGood	Number of successful Certificate Verify messages generated.
genCVNoKey	Number of unsuccessful CV message generations due to no key.
genCVConfigRSAFail	Number of unsuccessful CV message generations due to failed RSA configuration.
genCVRSAEncodeFail	Number of unsuccessful CV message generations due to failed RSA encryption.
processCVGood	Number of successful Certificate Verify messages processed.
processCVNoKey	Number of failed CV messages due to no key.
procssCVCfgRSAFail	Number of failed CV messages due to failed RSA configuration.
procssCVRSADecFail	Number of failed CV messages due to failed RSA decryption.
processCVFail	Number of failed CV messages.

Figure 30-14: Example output from the **show enco counter=user** command

ENCO User Interface Counter:			
startConfig	2	startReconfig	0
attachGood	2	attachFail	0
attachNoConfig	0	attachBadUserType	0
attachedInvalidChannel	0	attachedUnusedChannel	0
attachProcNotAvail	0		
reconfigInvalidChannel	0	reconfigUnusedChannel	0
reconfigNoConfig	0		
detachInvalidChannel	0	detachUnusedChannel	0
detachedInvalidChannel	0	detachedUnusedChannel	0
detachGood	0		
decodeInvalidChannel	0	decodeUnusedChannel	0
encodeInvalidChannel	0	encodeUnusedChannel	0
codedInvalidChannel	0	codedUnusedChannel	0
resetInvalidChannel	0	resetUnusedChannel	0
resetDoneInvalidChannel	0	resetDoneUnusedChannel	0
configBadMode	0	configBadUserType	0
configBadPktLength	0	configBadEncrType	0
configBadCompType	0	configBadHistoryMode	0
configBadCheckType	0		
discardInvalidChannel	0	discardUnusedChannel	0

Table 30-11: Parameters in output of the **show enco counter=user** command

Parameter	Meaning
startConfig	Number of channel configuration requests initiated.
startReconfig	Number of channel reconfiguration requests started.
attachGood	Number of successful channel attaches.
attachFail	Number of unsuccessful channel attaches.
attachNoConfig	Number of channel attach requests received with no configuration information.
attachBadUserType	Number of channel attach requests containing a bad user type.
attachedInvalidChannels	Number of channel attached events on invalid channels.
attachedUnusedChannel	Number of channel attached events on unused channels.
attachProcNotAvail	Number of attaches requesting a process while the process is unavailable.
reconfigInvalidChannel	Number of channel reconfigure requests on invalid channels.
reconfigUnusedChannel	Number of channel reconfigure requests on unused channels.
reconfigNoConfig	Number of channel reconfigure requests received with no configuration information.
detachInvalidChannel	Number of channel detach requests on nonexistent channels.
detachedInvalidChannel	Number of channel detached events on invalid channels.
detachedUnusedChannel	Number of channel detached events on unused channels.
detachGood	Number of successful channel detaches.
decodeInvalidChannel	Number of decode requests on nonexistent channels.
decodeUnusedChannel	Number of decode requests on unused channels.
encodeInvalidChannel	Number of encode requests on nonexistent channels.
encodeUnusedChannel	Number of encode requests on unused channels.
codedInvalidChannel	Number of encode events on nonexistent channels.
codedUnusedChannel	Number of encode events on unused channels.
resetInvalidChannel	Number of channel reset requests on nonexistent channels.
resetUnusedChannel	Number of channel reset requests on unused channels.
resetDoneInvalidChannel	Number of channel reset requests on invalid channels.
resetDoneUnusedChannel	Number of channel reset requests on nonexistent channels.
configBadMode	Number of channel configuration requests containing a bad mode.
configBadUserType	Number of channel configuration requests containing a bad user type.
configBadPktLength	Number of channel configuration requests containing a bad packet length.
configBadEncrType	Number of channel configuration requests containing a bad encryption type.
configBadCompType	Number of channel configuration requests containing a bad compression type.

Table 30-11: Parameters in output of the **show enco counter=user** command

Parameter	Meaning
configBadHistoryMode	Number of channel configuration requests containing a bad history mode.
configBadCheckType	Number of channel configuration requests containing a check type.
discardInvalidChannel	Number of discarded jobs on invalid channels.
discardUnusedChannel	Number of discarded jobs on nonexistent channels.

Figure 30-15: Example output from the **show enco counter=util** command

ENCO Utility Counter:			
codeNullPacket	0	codeBadPacketPriority	0
codeBadPacketLength	0	codeBadConfig	0
actionSentEncode	0	actionSentDecode	0
configureGood	2	configureFail	0
encodeGood	0	decodeGood	2
encodeBad	0	decodeBad	0

Table 30-12: Parameters in output of the **show enco counter=util** command

Parameter	Meaning
codeNullPacket	Number of utility jobs where the packet to be processed was null.
codeBadPacketLength	Number of utility jobs where the packet to be processed had a bad packet length.
actionSentEncode	Number of encode jobs started.
configureGood	Number of successful attempts to configure the utility channel.
encodeGood	Number of completed encode jobs.
encodeBad	Number of failed encode jobs.
codeBadPacketPriority	Number of utility jobs where the packet to be processed had a bad priority.
codeBadConfig	Number of utility jobs where the configuration was invalid.
actionSentDecode	Number of decode jobs started.
configureFail	Number of failed attempts to configure the utility channel.
decodeGood	Number of completed decode jobs.
decodeBad	Number of failed decode jobs.

Related Commands [reset enco counter](#)
[show enco](#)

show enco debug

Syntax SHow ENCo DEBug

Description This command executes a specific sequence of show commands to produce output useful for debugging. The specific commands are:

- **show enco**
- **show enco channel**
- **show enco counters** for all possible parameters

This command requires a user with Security Officer privilege when the switch is in security mode.

Examples To display information useful for debugging, use the command:

```
sh enc deb
```

Related Commands [disable enco debugging](#)
[enable enco debugging](#)
[reset enco counter](#)
[show enco](#)
[show enco channel](#)
[show enco counter](#)

show enco key

Syntax SHow ENCo KEY [=key-id]

where *key-id* is a number from 0 to 65535

Description This command displays information about keys stored in the ENCO key memory. It requires a user with security officer privilege when the switch is in security mode.

If a key identification number is not specified, a summary of all keys stored in key memory is displayed ([Figure 30-16](#), [Table 30-13 on page 30-36](#)). If an identification number is specified, only the specified key is displayed ([Figure 30-17](#)).

Figure 30-16: Example output from the **show enco key** command

ID	Algorithm	Length	Digest	Description	Module	IP Address
0	RSA-PRIVATE	768	E300FFDF	ROUTER KEY	-	-
1	RSA-PRIVATE	512	DC5014A8	SSH Key	SSH	-
2	RSA-PUBLIC	1024	2A1596C9	CHCH router	-	192.168.2.1
3	RSA-PUBLIC	1024	9348B823	Pauls key	-	-
4	RSA-PUBLIC	512	C4A396A0	Carls Datafellow key	-	-
5	DES	64	5A6798BD	Man key for CHCH SA	-	192.168.2.1
6	GENERAL	768	D56D323F	Auth key for INV	-	-
7	3DES2KEY	128	45FE62AB	Man key for INV SA	-	192.168.3.1

Table 30-13: Parameters in output of the **show enco key** command

Parameter	Meaning
ID	Identification number for the key.
Algorithm	Encryption algorithm for which the key was created: DES 3DES2KEY 3DESINNER RSA-PRIVATE RSA-PUBLIC GENERAL
Length	Length of the key in bytes/bits.
Digest	MD5 digest of the key data.
Description	User-defined description for the key.
Module	Module associated with this key.
IP Address	IP address associated with this key.

Figure 30-17: Example output from the **show enco key** command for a specific DES key

```
ijn69p4v95e5qk  
0x425BCFBF55FEC9B8
```

Examples To display the list of all enco keys use the command:

```
sh enc key
```

To display a single DES key use the command:

```
sh enc key=1
```

Related Commands [create enco key](#)
[destroy enco key](#)
[set enco key](#)