ICSF

Enhanced Key Management for Crypto Assist Instructions

Contents

Overview

This document supports APAR OA08172.

Support Description

To enable more exploitation of the clear key DES instructions on the CPACF, ICSF will be enhanced to generate and format clear DES tokens that can be used in callable services and stored in the cryptographic key data set (CKDS). Clear key tokens on the CKDS can be referenced by labelname by the Symmetric Key Encipher (CSNBSYE and CSNBSYE1) and the Symmetric Key Decipher (CSNBSYD and CSNBSYD1) services. With clear key support on the CKDS, clear keys do not have to appear in application storage during use.

Requirements

Software requirements are:

- APAR OA08172
- FMID HCR770A or HCR770B
- OS/390 V2R10 or later

Hardware requirements are:

- z890 or z990 server
- CP Assist for Cryptographic Functions (CPACF) DES/TDES Enablement (feature code 3863)
- PCI X Cryptographic Coprocessor (feature code 0868)

Changed Callable Services

The callable services provided here in their entirety are from the current *ICSF Application Programmer's Guide*. Updates to the callable services are marked with an asterisk (*) so that you can quickly find the new support documentation.

- Key Record Read (CSNBKRR)

  The Key Record Read service will not return the token for a clear key. See "Key Record Read (CSNBKRR)" on page 5.
- Key Record Write (CSNBKRW)

  The Key Record Write service supports writing a clear key token with non-zero key values to the CKDS.
- Key Token Build (CSNBKTB)

  The Key Token Build service will allow creation of a clear key token from supplied key values. A new *key_type*, CLRDES, is required. See "Key Token Build (CSNBKTB)" on page 7.
- Symmetric Key Decipher (CSNBSYD and CSNBSYD1)

  These services will now support the specification of a key token or key labelname for the *key_identifier* parameter. For the DES algorithm only, new keyword KEYIDENT may be specified for the key rule keyword in the *rule_array* parameter. See "Symmetric Key Decipher (CSNBSYD and CSNBSYD1)" on page 17.
- Symmetric Key Encipher (CSNBSYE and CSNBSYE1)

These services will now support the specification of a key token or key labelname for the *key_identifier* parameter. For the DES algorithm only, new keyword KEYIDENT may be specified for the key rule keyword in the *rule_array* parameter. See “Symmetric Key Encipher (CSNBSYE and CSNBSYE1)” on page 25.

KGUP

KGUP will support the creation and maintenance of clear key tokens on the CKDS. See “KGUP Updates” on page 33.

KGUP TSO Panels

See “Panels” on page 34.

Sharing the CKDS

On systems sharing the CKDS without this support, it is highly recommended that you RACF-protect the labelname of the clear key tokens on the other systems. This will provide additional security for your installation.

Message Changes

Message CSFG0224 has changed.

The message text is: *keyword* SPECIFIED WITH TYPE *keytype*.

There is a mismatch between *keyword* and *keytype*. *Keyword* values can be NOCV or DES. If NOCV is specified, only key types EXPORTER or IMPORTER are allowed. If DES is specified, only key types EXPORTER, IMPORTER, or DATA are allowed.

If *keytype* CLRDES is specified, *keywords* CLEAR, OUTTYPE and TRANSKEY are not allowed.

New Reason Code for Return Code 4 (4)

Table 1 lists the new reason code.

Table 1. Reason Codes for Return Code 4 (4)

Reason Code Hex (Decimal)	Description
81E (2078)	The key was retrieved successfully, but for a clear key token, it is not returned to the caller.

New Reason Code for Return Code 8 (8)

Table 2 lists the new reason code.

Table 2. Reason Codes for Return Code 8 (8)

Reason Code Hex (Decimal)	Description
81F (2079)	An encrypted key token is not supported in the service.

Clear Key Token

See “Format of the Clear Key Token” on page 37.

Key Record Read (CSNBKRR)

Use the key record read callable service to copy an internal key token from the in-storage CKDS to application storage. Other cryptographic services can then use the copied key token directly. The key token can also be used as input to the token copying functions of key generate, key import, or secure key import services to create additional NOCV keys.

* If the internal key token is a clear key token, the token is not returned to the caller
* unless the caller is in supervisor state or system key. Otherwise, a return code 4,
* reason code X'81E' is returned.

Format

```
CALL CSNBKRR(
             return_code,
             reason_code,
             exit_data_len th,
             exit_data,
             key_label,
             key_token)
```

Parameters

return_code

Direction: Output	Type: Integer

The return code specifies the general result of the callable service. *ICSF Application Programmer's Guide* lists the return codes.

reason_code

Direction: Output	Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it indicating specific processing problems. *ICSF Application Programmer's Guide* lists the reason codes.

exit_data_length

Direction: Input/Output	Type: Integer

The length of the data that is passed to the installation exit. The length can be from X'00000000' to X'7FFFFFFF' (2 gigabytes). The data is identified in the *exit_data* parameter.

exit_data

Direction: Input/Output	Type: String

The data that is passed to the installation exit.

key_label

Direction: Input Type: Character string

The 64-byte label of a record in the in-storage CKDS. The internal key token in this record is returned to the caller.

key_token

Direction:

Key Token Build (CSNBKTB)

Use the key token build callable service to build an external or internal key token from information which you supply. The token can be used as input for the key generate and key part import callable services. You can specify a control vector or the service can build a control vector based upon the key type you specify and the control vector-related keywords in the rule array. ICSF supports the building of an internal key token with the key encrypted under a master key other than the current master key.

Note: CLR8-ENC or UKPT must be coded in *rule_array* when the KEYGENKY *key_type* is coded. When the SECMSG *key_type* is coded, either SMKEY or SMPIN must be specified in the *rule_array*.

* You can use this service to build internal clear DES tokens. You must use *key_type*
* CLRDES. CLRDES requires *rule_array* keyword INTERNAL, optional keyword
* KEYLN8/ KEYLN16/ KEYLN24, and a *key_value* parameter.

You can also use this service to update the DES or SYS-ENC markings in a supplied DATA, IMPORTER, or EXPORTER token and to build CCA key tokens for all key types ICSF supports.

Format

```
CALL CSNBKTB(
             return_code,
             reason_code,
             exit_data_len th,
             exit_data,
             key_token,
             key_type,
             rule_array_count,
             rule_array,
             key_value,
             master_key_version_number,
             key_re ister_number,
             secure_token,
             control_vector,
             initialization_vector,
             pad_character,
             crypto raphic_period_start,
             masterkey_verify_parm
```

Parameters

return_code

Direction: Output	Type: Integer

The return code specifies the general result of the callable service. *ICSF Application Programmer's Guide* lists the return codes.

reason_code

Direction: Output	Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. *ICSF Application Programmer's Guide* lists the reason codes.

exit_data_length

Direction: Input/Output Type: Integer

Reserved field.

exit_data

Direction: Input/Output Type: String

Reserved field.

key_token

Direction: Input/Output Type: String

If the following parameter *key_type* is TOKEN then this is a 64-byte internal token that is updated as specified in the *rule_array*. The internal token must be a DATA, IMPORTER or EXPORTER key type. Otherwise this field is an output-only field.

key_type

Direction: Input Type: String

An 8-byte field that specifies the type of key you want to build or the keyword TOKEN for updating a supplied token. If *key_type* is TOKEN, then the *key_token* field cannot contain a double- or triple-length DATA key token. No other keywords are valid. The TOKEN keyword indicates changing an internal token in the *key_token* parameter. A valid *key_type* indicates building a key token from the parameters specified.

Key type values for the Key Token Build callable service are: AKEK, CIPHER,
* CLRDES, CVARDEC, CVARENC, CVARPINE, CVARXCVL, CVARXCVR, DATA, DATAC, DATAM, DATAMV, DATAXLAT, DECIPHER, DKYGENKY, ENCIPHER, EXPORTER, IKEYXLAT, IMPORTER, IPINENC, KEYGENKY, MAC, MACVER, OKEYXLAT, OPINENC, PINGEN, PINVER, and SECMSG. *Key_type* USE-CV is used when a user-supplied control vector is specified. The USE-CV *key_type* specifies that the *key_type* should be obtained from the control vector specified in the *control_vector* parameter. The CV *rule_array* keyword should be specified if USE-CV is specified.

* CLRDES can only specify a clear DES token. Using CLRDES requires
* *rule_array* keyword INTERNAL, optional keyword KEYLN8/KEYLN16/KEYLN24,
* and a *key_value* parameter.

rule_array_count

Direction: Input Type: Integer

The number of keywords you supplied in the *rule_array* parameter.

rule_array

Direction: Input Type: String

One to four keywords that provide control information to the callable service. See Table 4 for a list. The keywords must be in contiguous storage with each of the keywords left-justified in its own 8-byte location and padded on the right with blanks. For any key type, there are no more than four valid *rule_array* values.

If you specify TOKEN for the *key_type*, then the only valid *rule_array* values are INTERNAL and DES or SYS-ENC. The Data Encryption Algorithm (see the table that follows) keyword has no default.

If you specify a *key_type* of DATA, IMPORTER or EXPORTER, the Data Encryption Algorithm selection keyword defaults to SYS-ENC. The other *rule_array* keywords do not apply.

* If you specify a *key_type* of CLRDES, then the only valid *rule_array* values are
* INTERNAL and optionally, KEYLN8/KEYLN16/KEYLN24.

Table 4. Keywords for Key Token Build Control Information

Keyword	Meaning
Token Type (required)	
EXTERNAL	Specifies an external key token.
INTERNAL	Specifies an internal key token.
Key Status (optional) — not valid for CLRDES	
KEY	This keyword indicates that the key token to build will contain an encrypted key. The *key_value* parameter identifies the field that contains the key.
NO-KEY	This keyword indicates that the key token to build will not contain a key. This is the default key status.
Data Encryption Algorithm (optional) — valid only for single-length DATA keys and KEKs.	
DES	Tolerated for compatibility reasons.
SYS-ENC	Tolerated for compatibility reasons.
CV on the Link Specification (optional) — valid only for IMPORTER and EXPORTER.	
CV-KEK	This keyword indicates marking the KEK as a CV KEK. The control vector is applied to the KEK before use in encrypting other keys. This is the default.
NOCV-KEK	This keyword indicates marking the KEK as a NOCV KEK. The control vector is not applied to the KEK before use in encrypting other keys. Services using NO-CV keys must be processed on the Cryptographic Coprocessor Feature.
CV (Status optional) — not valid for CLRDES	
CV	This keyword indicates to obtain the control vector from the variable identified by the *control_vector* parameter.
NO-CV	Default. This keyword indicates that the control vector is to be supplied based on the key type and the control vector related keywords.
Key Length Keywords (optional)	

Table 4. Keywords for Key Token Build Control Information (continued)

Keyword	Meaning
DOUBLE	Double-length or 16-byte key. Synonymous with KEYLN16. Not valid for CLRDES. **Note:** See Table 5 on page 12 for valid key types for these key length values.
KEYLN8	Single-length or 8-byte key. Default for CLRDES.
KEYLN16	Double-length or 16-byte key.
KEYLN24	Triple-length, 24-byte key valid only for a DATA key type.
MIXED	Double-length key. Indicates that the key can either be a replicated single-length key or a double-length key with two different 8–byte values. Not valid for CLRDES.
SINGLE	Single-length or 8-byte key. Synonymous with KEYLN8. Not valid for CLRDES.
Key Part Indicator (optional) — not valid for CLRDES	
KEY-PART	This token is to be used as input to the key part import service.
Control Vector Keywords. Specify one or more of the following (optional)	
See Table 5 on page 12 for the key-usage keywords that can be specified for a given key type.	
Master Key Verification Pattern (optional) — not valid for CLRDES	
MKVP	This keyword indicates that the *key_value* is enciphered under the master key which corresponds to the master key verification pattern specified in the *masterkey_verify_parm* parameter. If this keyword is not specified, the key contained in the *key_value* field must be enciphered under the current master key.

key_value

Direction: Input   Type: String

If you use the KEY keyword, this parameter is a 16-byte string that contains the encrypted key value. Single-length keys must be left-justified in the field and padded on the right with X'00'. If you are building a triple-length DATA key, this parameter is a 24-byte string containing the encrypted key value. If you supply an encrypted key value and also specify INTERNAL, the service will check for the presence of the MKVP keyword. If MKVP is present, the service will assume the *key_value* is enciphered under the master key which corresponds to the master key verification pattern specified in the *masterkey_verify_parm* parameter, and will place the key into the internal token along with the verification pattern from the *masterkey_verify_parm* parameter. If MKVP is not specified, ICSF assumes the key is enciphered under the current host master key and places the key into an internal token along with the verification pattern for the current master key. In this case, the application must ensure that the master key has not changed since the key was generated or imported to this system. Otherwise, use of this parameter is not recommended.

For *key_type* CLRDES, this field is required to contain the clear key value. For KEYLN8, keys must be left-justified in the field and padded on the right with X'00'. KEYLN16 and KEYLN24 are also valid. For KEYLN24, this is a 24 byte field.

master_key_version_number

Direction: Input Type: Integer

This field is examined only if the KEY keyword is specified, in which case, this field must be zero. If the KEY and INTERNAL keywords are specified in *rule_array*, the service will check for the existence of the MKVP rule array keyword. If MKVP is specified, the service will make use of the last parameter specified (*masterkey_verify_parm*). The service assumes the key provided by the *key_value* parameter is enciphered under the corresponding master key and will place the key into the internal token along with the verification pattern from the *masterkey_verify_parm* parameter.

key_register_number

Direction: Input Type: Integer

This field is ignored.

secure_token

Direction: Input Type: String

This field is ignored.

control_vector

Direction: Input Type: String

A pointer to a 16 byte string variable. If this parameter is specified, and you use the CV rule array keyword, the variable is copied to the control vector field of the key token.

initialization_vector

Direction: Input Type: String

This field is ignored.

pad_character

Direction: Input Type: Integer

The only allowed value for key types MAC and MACVER is 0. This field is ignored for all other key types.

cryptographic_period_start

Direction: Input Type: String

This field is ignored.

masterkey_verify_parm

Direction: Input Type: String

A pointer to an 8-byte string variable. The value is inserted into the key token when you specify both the KEY and INTERNAL keywords in rule array.

Usage Notes

No pre- or post-processing or security exits are enabled for this service. No RACF checking is done, and no calls to RACF are issued when this service is used.

You can use this service to create skeleton key tokens with the desired data encryption algorithm bits for use in some key management services to override the default system specifications.

- If you are running with the Cryptographic Coprocessor Feature and need to generate operational AKEKs, use *key_type* of TOKEN and provide a skeleton AKEK key token as the *generated_key_identifier_1* into the key generate service.
- If you are running with the Cryptographic Coprocessor Feature, the KEY-PART AKEK key token can also be used as input to key part import service.
- To create an internal token with a specified KEY value, ICSF needs to supply a valid master key verification pattern (MKVP).

NOCV keyword is only supported for the standard IMPORTERs and EXPORTERs with the default CVs.

The following illustrates the key type and key usage keywords that can be combined in the Control Vector Generate and Key Token Build callable services to create a control vector.

Table 5. Control Vector Generate and Key Token Build Control Vector Keyword Combinations

Key Type	Key Usage			
	Default keys are indicated in bold. **A key usage keyword is required for the KEYGENKY key type.** *** All keywords in the list are defaults unless one or more keywords in the list are specified.** **** The NOOFFSET keyword is only valid if NO-SPEC, IBM-PIN, GBP-PIN, or the default (NO-SPEC) is specified.**			
Notes:	Default keys are indicated in bold. CLR8-ENC and/or UKPT must be specified for the KEYGENKY key type - SMKEY or SMPIN must be specified for the SECMSG key type * All keywords in the list are defaults unless one or more keywords in the list are specified. ** The NOOFFSET keyword is only valid if NO-SPEC, IBM-PIN, GBP-PIN, or the default (NO-SPEC) is specified.			
DATA		**SINGLE** KEYLN8 MIXED DOUBLE KEYLN16 KEYLN24	**XPORT-OK** NO-XPORT	KEY-PART
CIPHER ENCIPHER DECIPHER MAC MACVER		**SINGLE** KEYLN8 MIXED DOUBLE KEYLN16	**XPORT-OK** NO-XPORT	KEY-PART

Table 5. Control Vector Generate and Key Token Build Control Vector Keyword Combinations (continued)

Key Type	**Key Usage** **Default keys are indicated in bold.** **A key usage keyword is required for the KEYGENKY key type.** *** All keywords in the list are defaults unless one or more keywords in the list are specified.** **** The NOOFFSET keyword is only valid if NO-SPEC, IBM-PIN, GBP-PIN, or the default (NO-SPEC) is specified.**						
DATAXLAT CVARPINE CVARENC CVARDEC CVARXCVL CVARXCVR					**SINGLE** KEYLN8	**XPORT-OK** NO-XPORT	KEY-PART
DATAC DATAM DATAMV					**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART
KEYGENKY	CLR8-ENC UKPT				**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART
DKYGENKY	DDATA DMAC DMV DIMP DEXP DPVR DMKEY DMPIN DALL	DKYL0 DKYL1 DKYL2 DKYL3 DKYL4 DKYL5 DKYL6 DKYL7			**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART
SECMSG	SMKEY SMPIN				**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART
IKEYXLAT OKEYXLAT				**ANY** NOT-KEK DATA PIN LMTD-KEK	**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART
IMPORTER	OPIM* IMEX* IMIM* IMPORT*	XLATE		**ANY** NOT-KEK DATA PIN LMTD-KEK	**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART
EXPORTER	OPEX* IMEX* EXEX* EXPORT*	XLATE		**ANY** NOT-KEK DATA PIN LMTD-KEK	**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART
PINVER			**NO-SPEC**** IBM-PIN** GBP-PIN** IBM-PINO GBP-PINO VISA-PVV INBK-PIN	NOOFFSET	**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART

Table 5. Control Vector Generate and Key Token Build Control Vector Keyword Combinations (continued)

Key Type	**Key Usage** **Default keys are indicated in bold.** **A key usage keyword is required for the KEYGENKY key type.** *** All keywords in the list are defaults unless one or more keywords in the list are specified.** **** The NOOFFSET keyword is only valid if NO-SPEC, IBM-PIN, GBP-PIN, or the default (NO-SPEC) is specified.**					
PINGEN	CPINGEN* CPINGENA* EPINGENA* EPINGEN* EPINVER*	**NO-SPEC**** IBM-PIN** GBP-PIN** IBM-PINO GBP-PINO VISA-PVV INBK-PIN	NOOFFSET	**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART
IPINENC	CPINGENA* EPINVER* REFORMAT* TRANSLAT*			**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART
OPINENC	CPINENC* EPINGEN* REFORMAT* TRANSLAT*			**DOUBLE** KEYLN16 MIXED	**XPORT-OK** NO-XPORT	KEY-PART

Related Information

Attention: CDMF is no longer supported.

The ICSF key token build callable service provides a subset of the parameters and keywords available with the Transaction Security System key token build verb.

The following key types are not supported: ADATA, AMAC, CIPHERXI, CIPHERXL, CIPHERXO, UKPTBASE.

The following rule array keywords are not supported: ACTIVE, ADAPTER, CARD, CBC, CLEAR-IV, CUSP, INACTIVE, IPS, KEY-REF, MACLEN4, MACLEN6, MACLEN8, NO-IV, READER, X9.2, X9.9-1.

The *master_key_verification_number* parameter has been replaced by the *master_key_version_number* parameter. The *master_key_version_number* parameter is examined only if the KEY keyword is specified, and in this case must be zero. If KEY and INTERNAL are both specified in the rule array, the service will check for the existence of a new optional rule array keyword, MKVP. If MKVP is specified, the service will make use of the last parameter specified. Currently, this is called *masterkey_verify_parm* and is always ignored. It will now be used to contain a master key verification pattern if MKVP is specified in the *rule_array*. The service assumes the key provided by the *key_value* parameter is enciphered under the corresponding master key and will place the key into the internal token along with the verification pattern from the *masterkey_verify_parm* parameter.

The *key_register_number*, *secure_token*, and *initialization_vector* parameters are ignored.

The *pad_character* parameter must have a value of zero.

The following table lists the required cryptographic hardware for each server type and describes restrictions for this callable service.

Table 6. Key token build required hardware

Server	**Required cryptographic hardware**	**Restrictions**
S/390 G6 Enterprise Server	None.	
IBM @server zSeries 800 IBM @server zSeries 900	None.	
IBM @server zSeries 990 IBM @server zSeries 890	None.	

Symmetric Key Decipher (CSNBSYD and CSNBSYD1)

Use the symmetric key decipher callable service to decipher data in an address space or a data space using the cipher block chaining or electronic code book modes. ICSF supports the following processing rules to decipher data. You choose the type of processing rule that the decipher callable service should use for block chaining.

Processing Rule	**Purpose**
ANSI X9.23	For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, but the plaintext will be 1 to 8 bytes shorter than the ciphertext.
CBC	For cipher block chaining. The ciphertext must be an exact multiple of 8 bytes, and the plaintext will have the same length.
CUSP	For cipher block chaining, but the ciphertext can be of any length. The plaintext will be the same length as the ciphertext.
ECB	Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm.
IPS	For cipher block chaining, but the ciphertext can be of any length. The plaintext will be the same length as the ciphertext.

The Advanced Encryption Standard (AES) and DES (Data Encryption Standard) are supported. AES encryption uses a 128-, 192-, or 256-bit key. The CBC and ECB modes are supported. Due to export regulations, AES encryption may not be available on your system.

This service supports both electronic code book (ECB) and cipher block chaining (CBC) modes. The CBC mode of operation uses an initial chaining vector (ICV) in its processing. The ICV is exclusive ORed with the first block of plaintext after the decryption step, and thereafter, each block of ciphertext is exclusive ORed with the next block of plaintext after decryption, and so on.

Cipher block chaining also produces a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next encipher call. This results in record chaining.

The electronic code book mode does not use the initial chaining vector.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the *key_identifier* parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

* For DES, the key may be specified as a clear key value or the *key_identifier* of a
* clear key token or labelname in the CKDS.

Choosing Between CSNBSYD and CSNBSYD1

CSNBSYD and CSNBSYD1 provide identical functions. When choosing which service to use, consider the following:

- **CSNBSYD** requires the ciphertext and plaintext to reside in the caller's primary address space. Also, a program using CSNBSYD adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- **CSNBSYD1** allows the ciphertext and plaintext to reside either in the caller's primary address space or in a data space. This can allow you to decipher more data with one call. However, a program using CSNBSYD1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified before it can run with other cryptographic products that follow this programming interface.

  For CSNBSYD1, *cipher_text_id* and *clear_text_id* are access list entry token (ALET) parameters of the data spaces containing the ciphertext and plaintext.

Format

```
CALL CSNBSYD(
             return_code,
             reason_code,
             exit_data_len th,
             exit_data,
             rule_array_count,
             rule_array,
             key_len th,
             key_identifier,
             key_parms_len th,
             key_parms,
             block_size,
             initialization_vector_len th,
             initialization_vector,
             chain_data_len th,
             chain_data,
             cipher_text_len th,
             cipher_text,
             clear_text_len th,
             clear_text,
             optional_data_len th,
             optional_data)
```

```
CALL CSNBSYD1(
             return_code,
             reason_code,
             exit_data_len th,
             exit_data,
             rule_array_count,
             rule_array,
             key_len th,
             key_identifier,
             key_parms_len th,
             key_parms,
             block_size,
             initialization_vector_len th,
             initialization_vector,
             chain_data_len th,
             chain_data,
             cipher_text_len th,
             cipher_text,
             clear_text_len th,
             clear_text,
             optional_data_len th,
             optional_data
             cipher_text_id
             clear_text_id)
```

Parameters

return_code

Direction: Output Type: Integer

The return code specifies the general result of the callable service. *ICSF Application Programmer's Guide* lists the return codes.

reason_code

Direction: Output Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. *ICSF Application Programmer's Guide* lists the reason codes.

exit_data_length

Direction: Ignored Type: Integer

Reserved field.

exit_data

Direction: Ignored Type: String

Reserved field.

rule_array_count

Direction: Input Type: Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 1, 2, 3 or 4.

rule_array

Direction: Input Type: String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

Table 7. Symmetric Key Decipher Rule Array Keywords

Keyword	Meaning
Algorithm (required)	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes. The *chain_data* field must be at least 32 bytes in length. The OCV is the first 16 bytes in the *chain_data*. The supported processing rules for AES are CBC and ECB.
DES	Specifies that the Data Encryption Standard (DES) algorithm is to be used. The algorithm, DES or TDES, will be determined from the length of the key supplied. The key length may be 8, 16, or 24. The block size is 8 bytes. The *chain_data* field must be at least 16 bytes in length. The OCV is the first eight bytes in the *chain_data*. The processing rules supported for DES are CBC, ECB, X9.23, CUSP and IPS.
Processing Rule (optional)	
CBC	Performs cipher block chaining. The text length must be a multiple of the block size for the specified algorithm. CBC is the default value.
CUSP	CBC mode (cipher block chaining) that is compatible with IBM's CUSP and PCF products. Input text may be any length.
ECB	Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm.
IPS	CBC mode (cipher block chaining) that is compatible with IBM's IPS product. Input text may be any length.
X9.23	CBC mode (cipher block chaining) for 1 to 8 bytes of padding dropped from the output clear text.
Key Rule (optional)	
KEY-CLR	This specifies that the key parameter contains a clear key value. KEY-CLR is the default value.
* KEYIDENT	This specifies that the *key_identifier* field will be an internal clear token or the labelname of a key in the CKDS. Normal CKDS labelname syntax is required. Valid only with DES.
ICV Selection (optional)	
INITIAL	This specifies taking the initialization vector from the *initialization_vector* parameter. INITIAL is the default value.

Table 7. Symmetric Key Decipher Rule Array Keywords (continued)

Keyword	Meaning
CONTINUE	This specifies taking the initialization vector from the output chaining vector contained in the work area to which the *chain_data* parameter points. CONTINUE is valid for processing rules CBC, IPS, and CUSP only.

key_length

Direction: Input Type: Integer

The length of the key parameter. For clear keys, the length is in bytes and includes only the value of the key. The maximum size is 256 bytes.

* For the KEYIDENT keyword, this parameter value must be 64.

key_identifier

Direction: Input Type: String

For the KEY-CLR keyword, this specifies the cipher key. The parameter must be left justified.

* For the KEYIDENT keyword, this specifies an internal clear token or the
* labelname of a key in the CKDS. Normal CKDS labelname syntax is required.
* KEYIDENT is only valid with DES.

key_parms_length

Direction: Ignored Type: Integer

The length of the *key_parms* parameter. The maximum size is 256 bytes.

key_parms

Direction: Ignored Type: String

This parameter contains key-related parameters specific to the encryption algorithm.

block_size

Direction: Input Type: Integer

This parameter contains the processing size of the text block in bytes. This value will be algorithm specific. Be sure to specify the same block size as used to encipher the text.

initialization_vector_length

Direction: Input Type: Integer

The length of the *initialization_vector* parameter. The length should be equal to the block length for the algorithm specified.

initialization_vector

Direction: Input Type: String

This initialization chaining value for CBC encryption. You must use the same ICV that was used to encipher the data.

chain_data_length

Direction: Input/Output Type: Integer

The length of the *chain_data* parameter. On output, the actual length of the chaining vector will be stored in the parameter.

chain_data

Direction: Input/Output Type: String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The direction is output if the ICV selection keyword is INITIAL.

The mapping of the *chain_data* depends on the algorithm specified. For AES, the *chain_data* field must be at least 32 bytes in length. The OCV is in the first 16 bytes in the *chain_data*. For DES, *chain_data* field must be at least 16 bytes in length.

cipher_text_length

Direction: Input Type: Integer

The length of the cipher text. A zero value in the *clear_text_length* parameter is not valid. The length must be a multiple of the algorithm block size.

cipher_text

Direction: Input Type: String

The text to be deciphered.

clear_text_length

Direction: Input/Output Type: Integer

On input, this parameter specifies the size of the storage pointed to by the *clear_text* parameter. On output, this parameter has the actual length of the text stored in the *clear_text* parameter.

clear_text

Direction: Output Type: String

The deciphered text the service returns.

optional_data_length

Direction: Ignored Type: Integer

The length of the *optional_data* parameter.

optional_data

Direction: Ignored	Type: String

Optional data required by a specified algorithm.

cipher_text_id

Direction: Input	Type: Integer

For CSNBSYD1 only, the ALET of the ciphertext to be deciphered.

clear_text_id

Direction: Input	Type: Integer

For CSNBSYD1 only, the ALET of the clear text supplied by the application.

Usage Notes

- No pre- or post-processing exits are enabled for this service.
- No SAF authorization check is made.
- The master keys need not be loaded to use this service.
- The AES algorithm is implemented in the software.
- AES has the same availability restrictions as triple-DES.
- This service will fail if execution would cause destructive overlay of the *cipher_text* field.

Table 8. Symmetric Key Decipher required hardware

Server	**Required cryptographic hardware**	**Restrictions**
S/390 G6 Enterprise Server	Cryptographic Coprocessor Feature	DES keyword is not supported.
IBM @server zSeries 800 IBM @server zSeries 900	Cryptographic Coprocessor Feature	DES keyword is not supported.
IBM @server zSeries 990 IBM @server zSeries 890	CP Assist for Cryptographic Functions	

Related Information

You **cannot** overlap the plaintext and ciphertext fields. For example:

```
pppppp
     cccccc  is not supported.

cccccc
     pppppp  is not supported.
```

```
pppppppcccccc is supported.

P represents the plaintext and c represents the ciphertext.
```

On z990 systems, the PCIXCC will support non destructive overlap. For example:

```
pppppp
     cccccc  is supported.
```

ICSF Application Programmer's Guide discusses the cipher processing rules.

Symmetric Key Encipher (CSNBSYE and CSNBSYE1)

Use the symmetric key encipher callable service to encipher data in an address space or a data space using the cipher block chaining or electronic code book modes. ICSF supports the following processing rules to encipher data. You choose the type of processing rule that the encipher callable service should use for the block chaining.

Processing Rule	Purpose
ANSI X9.23	For block chaining not necessarily in exact multiples of 8 bytes. This process rule pads the plaintext so that ciphertext produced is an exact multiple of 8 bytes.
CBC	For block chaining in exact multiples of 8 bytes.
CUSP	For block chaining not necessarily in exact multiples of 8 bytes. The ciphertext will be the same length as the plaintext.
ECB	Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm.
IPS	For block chaining not necessarily in exact multiples of 8 bytes. The ciphertext will be the same length as the plaintext.

The Advanced Encryption Standard (AES) and DES (Data Encryption Standard) are supported. AES encryption uses a 128-, 192-, or 256-bit key. The CBC and ECB modes are supported. Due to export regulations, AES encryption may not be available on your system.

This service supports both electronic code book (ECB) and cipher block chaining (CBC) modes. The CBC mode of operation uses an initial chaining vector (ICV) in its processing. The ICV is exclusive ORed with the first block of plaintext before the encryption step, and thereafter, the block of ciphertext just produced is exclusive ORed with the next block of plaintext, and so on. This disguises any pattern that may exist in the plaintext.

Cipher block chaining also produces a resulting chaining value called the output chaining vector (OCV). The application can pass the OCV as the ICV in the next encipher call. This results in record chaining.

The electronic code book mode does not use the initial chaining vector.

The selection between single-DES decryption mode and triple-DES decryption mode is controlled by the length of the key supplied in the *key_identifier* parameter. If a single-length key is supplied, single-DES decryption is performed. If a double-length or triple-length key is supplied, triple-DES decryption is performed.

* For DES, the key may be specified as a clear key value or the *key_identifier* of a
* clear key token or labelname in the CKDS.

Choosing between CSNBSYE and CSNBSYE1

CSNBSYE and CSNBSYE1 provide identical functions. When choosing which service to use, consider the following:

- **CSNBSYE** requires the cleartext and ciphertext to reside in the caller's primary address space. Also, a program using CSNBSYE adheres to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface.
- **CSNBSYE1** allows the cleartext and ciphertext to reside either in the caller's primary address space or in a data space. This can allow you to encipher more data with one call. However, a program using CSNBSYE1 does not adhere to the IBM Common Cryptographic Architecture: Cryptographic Application Programming Interface, and may need to be modified before it can run with other cryptographic products that follow this programming interface.

  For CSNBSYE1, *clear_text_id* and *cipher_text_id* are access list entry token (ALET) parameters of the data spaces containing the cleartext and ciphertext.

Format

```
CALL CSNBSYE(
            return_code,
            reason_code,
            exit_data_len th,
            exit_data,
            rule_array_count,
            rule_array,
            key_len th,
            key_identifier,
            key_parms_len th,
            key_parms,
            block_size,
            initialization_vector_len th,
            initialization_vector,
            chain_data_len th,
            chain_data,
            clear_text_len th,
            clear_text,
            cipher_text_len th,
            cipher_text,
            optional_data_len th,
            optional_data)
```

```
CALL CSNBSYE1(
             return_code,
             reason_code,
             exit_data_len th,
             exit_data,
             rule_array_count,
             rule_array,
             key_len th,
             key_identifier,
             key_parms_len th,
             key_parms,
             block_size,
             initialization_vector_len th,
             initialization_vector,
             chain_data_len th,
             chain_data,
             clear_text_len th,
             clear_text,
             cipher_text_len th,
             cipher_text,
             optional_data_len th,
             optional_data
             clear_text_id
             cipher_text_id)
```

Parameters

return_code

Direction: Output	Type: Integer

The return code specifies the general result of the callable service. *ICSF Application Programmer's Guide* lists the return codes.

reason_code

Direction: Output	Type: Integer

The reason code specifies the result of the callable service that is returned to the application program. Each return code has different reason codes assigned to it that indicate specific processing problems. *ICSF Application Programmer's Guide* lists the reason codes.

exit_data_length

Direction: Ignored	Type: Integer

Reserved field.

exit_data

Direction: Ignored	Type: String

Reserved field.

rule_array_count

Direction: Input	Type: Integer

The number of keywords you supplied in the *rule_array* parameter. The value may be 1, 2, 3 or 4.

rule_array

Direction: Input	Type: String

An array of 8-byte keywords providing the processing control information. The keywords must be in contiguous storage, left-justified and padded on the right with blanks.

Table 9. Symmetric Key Encipher Rule Array Keywords

Keyword	Meaning
Algorithm (required)	
AES	Specifies that the Advanced Encryption Standard (AES) algorithm is to be used. On systems that contain a Cryptographic Coprocessor Feature, AES is the only algorithm that is supported. The block size is 16 bytes. The key length may be 16, 24, or 32 bytes. The *chain_data* field must be at least 32 bytes in length. The OCV is the first 16 bytes in the *chain_data*.The supported processing rules for AES are CBC and ECB.
DES	Specifies that the Data Encryption Standard (DES) algorithm is to be used. The algorithm, DES or TDES, will be determined from the length of the key supplied. The key length may be 8, 16, or 24. The block size is 8 bytes. The *chain_data* field must be at least 16 bytes in length. The OCV is the first eight bytes in the *chain_data*. The processing rules supported for DES are CBC, ECB, X9.23, CUSP and IPS.
Processing Rule (optional)	
CBC	Performs cipher block chaining. The text length must be a multiple of the block size for the specified algorithm. CBC is the default value.
CUSP	CBC mode (cipher block chaining) that is compatible with IBM's CUSP and PCF products. Input text may be any length.
ECB	Performs electronic code book encryption. The text length must be a multiple of the block size for the specified algorithm.
IPS	CBC mode (cipher block chaining) that is compatible with IBM's IPS product. Input text may be any length.
X9.23	CBC mode (cipher block chaining) for 1 to 8 bytes of padding added according to ANSI X9.23. Input text may be any length.
Key Rule (optional)	
KEY-CLR	This specifies that the key parameter contains a clear key value. KEY-CLR is the default.
* KEYIDENT	This specifies that the key parameter contains that the *key_identifier* field will be an internal clear token or the labelname of a key in the CKDS. Normal CKDS labelname syntax is required. Only valid with DES.
ICV Selection (optional)	

Table 9. Symmetric Key Encipher Rule Array Keywords (continued)

Keyword	Meaning
INITIAL	This specifies taking the initialization vector from the *initialization_vector* parameter. INITIAL is the default value.
CONTINUE	This specifies taking the initialization vector from the output chaining vector contained in the work area to which the *chain_data* parameter points. CONTINUE is valid for processing rules CBC, IPS, and CUSP only.

key_length

Direction: Input | Type: Integer

The length of the key parameter. For clear keys, the length is in bytes and includes only the value of the key.

* For the KEYIDENT keyword, this parameter value must be 64.

key_identifier

Direction: Input | Type: String

For the KEY-CLR keyword, this specifies the cipher key. The parameter must be left justified.

* For the KEYIDENT keyword, this specifies a internal clear token or the
* labelname of a key in the CKDS. Normal CKDS labelname syntax is required.
* KEYIDENT is only valid with DES.

key_parms_length

Direction: Ignored | Type: Integer

The length of the *key_parms* parameter.

key_parms

Direction: Ignored | Type: String

This parameter contains key-related parameters specific to the encryption algorithm.

block_size

Direction: Input | Type: Integer

This parameter contains the processing size of the text block in bytes. This value will be algorithm specific.

initialization_vector_length

Direction: Input | Type: Integer

The length of the *initialization_vector* parameter. The length should be equal to the block length for the algorithm specified.

initialization_vector

Direction: Input	Type: String

This initialization chaining value for CBC encryption. You must use the same ICV to decipher the data.

chain_data_length

Direction: Input/Output	Type: Integer

The length of the *chain_data* parameter. On output, the actual length of the chaining vector will be stored in the parameter.

chain_data

Direction: Input/Output	Type: String

This field is used as a system work area for the chaining vector. Your application program must not change the data in this string. The chaining vector holds the output chaining vector from the caller.

The direction is output if the ICV selection keyword is INITIAL.

The mapping of the *chain_data* depends on the algorithm specified. For AES, the *chain_data* field must be at least 32 bytes in length. The OCV is in the first 16 bytes in the *chain_data*. For DES, the *chain_data* field must be at least 16 bytes in length.

clear_text_length

Direction: Input	Type: Integer

The length of the clear text. A zero value in the *clear_text_length* parameter is not valid. The length must be a multiple of the algorithm block size.

clear_text

Direction: Input	Type: String

The text to be enciphered.

cipher_text_length

Direction: Input/Output	Type: Integer

On input, this parameter specifies the size of the storage pointed to by the *cipher_text* parameter. On output, this parameter has the actual length of the text stored in the buffer addressed by the *cipher_text* parameter.

cipher_text

Direction: Output	Type: String

The enciphered text the service returns.

optional_data_length

Direction: Ignored	Type: Integer

The length of the *optional_data* parameter.

optional_data

Direction: Ignored	Type: String

Optional data required by a specified algorithm.

clear_text_id

Direction: Input	Type: Integer

For CSNBSYE1 only, the ALET of the clear text to be enciphered.

cipher_text_id

Direction: Input	Type: Integer

For CSNBSYE1 only, the ALET of the ciphertext that the application supplied.

Usage Notes

- No pre- or post-processing exits are enabled for this service.
- No SAF authorization check is made.
- The master keys need not be loaded to use this service.
- The AES algorithm is implemented in the software.
- AES has the same availability restrictions as triple-DES.
- This service will fail if execution would cause destructive overlay of the *clear_text* field.

Table 10. Symmetric Key Encipher required hardware

Server	**Required cryptographic hardware**	**Restrictions**
S/390 G6 Enterprise Server	Cryptographic Coprocessor Feature	DES keyword is not supported.
IBM @server zSeries 800 IBM @server zSeries 900	Cryptographic Coprocessor Feature	DES keyword is not supported.
IBM @server zSeries 990 IBM @server zSeries 890	CP Assist for Cryptographic Functions	

Related Information

You **cannot** overlap the plaintext and ciphertext fields. For example:

```
pppppp
     cccccc  is not supported.

cccccc
     pppppp  is not supported.
```

```
ppppppcccccc is supported.

P represents the plaintext and c represents the ciphertext.
```

On z990 systems, the PCIXCC will support non destructive overlap. For example:

```
cccccc
     pppppp  is supported.
```

The method used to produce the OCV is the same with the CBC and X9.23 processing rules. However, that method is different from the method used by the CUSP and IPS processing rules.

ICSF Application Programmer's Guide discusses the cipher processing rules.

KGUP Updates

The Key Generation Utility Program (KGUP) will support the creation and maintenance of clear key tokens in the CKDS. The new key type keyword CLRDES will be used to refer to clear DES key tokens. The ADD and UPDATE control statements will create or update clear key tokens.

The keywords that may be specified with TYPE(CLRDES) are LABEL or RANGE, LENGTH, and KEY. LABEL or RANGE is required. LENGTH is optional, the default is 8 (the SINGLE keyword should not be allowed for key types that may be single or double length). KEY is optional and if not specified, KGUP will generate a key of requested length. All other keywords are not supported with key type CLRDES.

Clear key values will not be echoed in any output data set. TYPE(CLRDES) can also be used on the RENAME and DELETE control statements.

Examples of Control Statements

For the ADD control statements, KGUP checks that the key label with a key type of CLRDES does not already exist in the CKDS. It also checks that there are no DATA, DATAXLAT, DATAM, DATAMV, MAC, MACVER, or NULL key entries with that label. Each of these key types require a unique label. If the key entry already exists, KGUP stops processing the control statement.

If the label does not exist, KGUP will create the key label in the CKDS.

Example 1 – ADD control statement with CLRDES keyword

This example shows a control statement that adds a CLRDES key label to the CKDS with a random 8 byte key.

```
ADD TYPE(CLRDES) LENGTH(8), LAB(CLRDES.KEYLN8)
```

Example 2 – ADD control statement to add a group of CLRDES key labels

This example shows a control statement that adds a group of CLRDES key labels to the CKDS. Key value is generated.

```
ADD TYPE(CLRDES) LENGTH(8),LAB(A.CLRDES.KEYLN8,B.CLRDES.KEYLN8,C.CLRDES.KEYLN8)
```

Example 3 – ADD control statement to add a group of CLRDES key labels

This example shows a control statement that adds a group of CLRDES key labels. The clear key value is specified.

```
ADD TYPE(CLRDES),KEY(2C2C2C2C2C2C2C2C,1616161616161616),
LAB(X.CLRDES.KEYLN16,Y.CLRDES.KEYLN16,Z.CLRDES.KEYLN16)
```

Example 4 – ADD control statement to add a range of CLRDES key labels

This example shows a control statement that adds a range of CLRDES key labels. A different key value is generated for each key label.

```
ADD TYPE(CLRDES) LENGTH(24),RAN(CLRDES.KEYLN24.KEY1,CLRDES.KEYLN24.KEY3)
```

Example 5 – UPDATE control statement with CLRDES keyword

This example shows a control statement that changes a CLRDES key label.

```
UPDATE TYPE(CLRDES),KEY(4343434343434343),LAB(CLRDES.KEYLN8)
```

Example 6 – UPDATE control statement with CLRDES keyword

This example shows a control statement that changes a range of CLRDES key labels.

```
UPDATE TYPE(CLRDES) LENGTH(16),RAN(CLRDES.KEY1,CLRDES.KEY3)
```

Example 7 – DELETE control statement with CLRDES keyword

This example shows a control statement that deletes a CLRDES key label.

```
DELETE TYPE(CLRDES),LAB(CLRDES.KEYLN24)
```

Example 8 – DELETE control statement to delete a group of CLRDES key labels

This example shows a control statement that deletes a group of CLRDES key labels.

```
DELETE TYPE(CLRDES),LAB(A.KEYLN16,B.KEYLN16,C.KEYLN16)
```

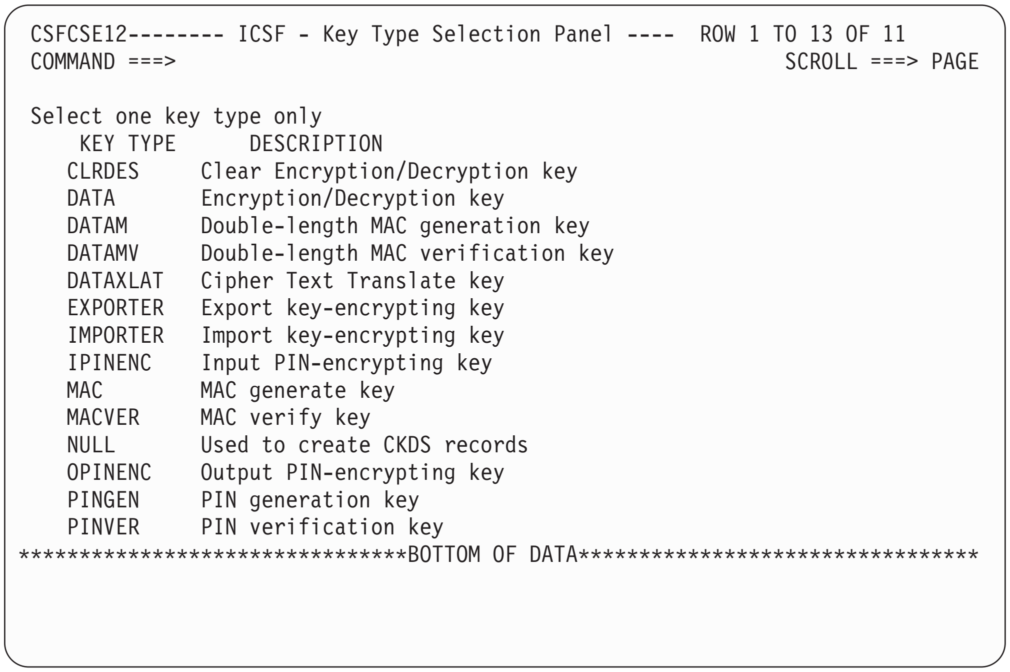
Example 9 – RENAME Control Statement with CLRDES Keyword

This example shows a control statement that renames a CLRDES key label.

```
RENAME TYPE(CLRDES),LAB(CLRDES.KEYLN16,CLRDES.DOUBLE.LENGTH.KEY)
```

Panels

This is the changed panel.

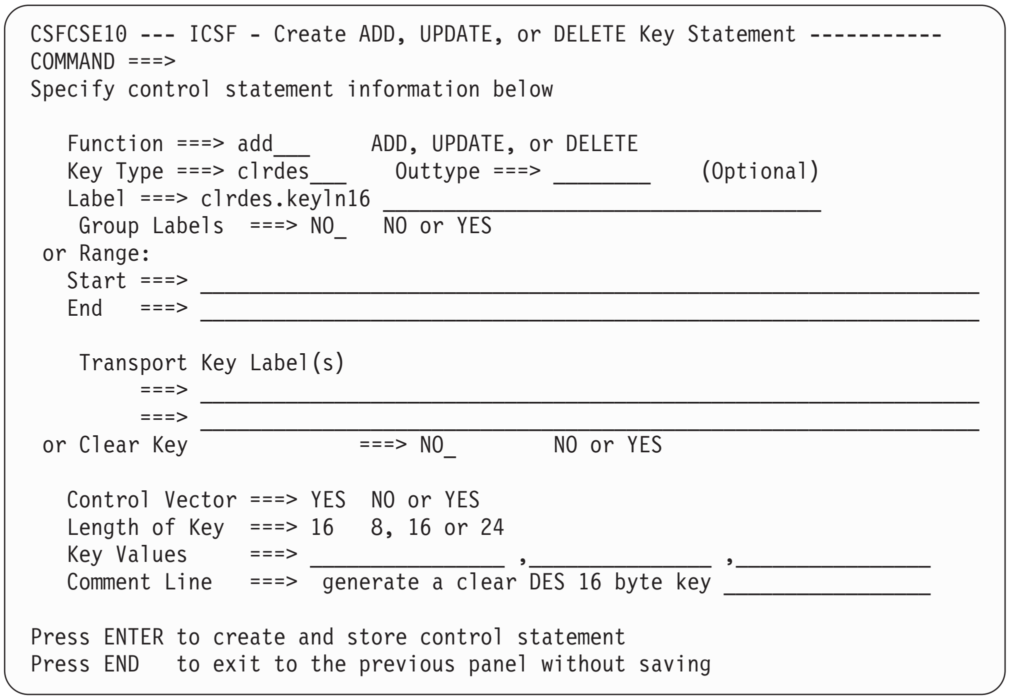


```
 CSFCSE12-------- ICSF - Key Type Selection Panel ----  ROW 1 TO 13 OF 11
 COMMAND ===>                                                SCROLL ===> PAGE

 Select one key type only
     KEY TYPE      DESCRIPTION
    CLRDES     Clear Encryption/Decryption key
    DATA       Encryption/Decryption key
    DATAM      Double-length MAC generation key
    DATAMV     Double-length MAC verification key
    DATAXLAT   Cipher Text Translate key
    EXPORTER   Export key-encrypting key
    IMPORTER   Import key-encrypting key
    IPINENC    Input PIN-encrypting key
    MAC        MAC generate key
    MACVER     MAC verify key
    NULL       Used to create CKDS records
    OPINENC    Output PIN-encrypting key
    PINGEN     PIN generation key
    PINVER     PIN verification key
********************************BOTTOM OF DATA*********************************
```

Figure 1. Selecting a Key on the Key Type Selection Panel

This is an example of the filled in panel.


```
CSFCSE10 --- ICSF - Create ADD, UPDATE, or DELETE Key Statement -----------
COMMAND ===>
Specify control statement information below

   Function ===> add___     ADD, UPDATE, or DELETE
   Key Type ===> clrdes___    Outtype ===> ________    (Optional)
   Label ===> clrdes.keyln16 ____________________________________
    Group Labels  ===> NO_   NO or YES
 or Range:
   Start ===> ________________________________________________________________
   End   ===> ________________________________________________________________

    Transport Key Label(s)
         ===> ________________________________________________________________
         ===> ________________________________________________________________
 or Clear Key              ===> NO_        NO or YES

   Control Vector ===> YES  NO or YES
   Length of Key  ===> 16   8, 16 or 24
   Key Values     ===> ________________ ,_______________ ,________________
   Comment Line   ===>  generate a clear DES 16 byte key _________________

Press ENTER to create and store control statement
Press END   to exit to the previous panel without saving
```

Figure 2. Sample ADD Key Statement Panel

Format of the Clear Key Token

Table 11 shows the format for a clear internal key token.

Table 11. Internal Clear Key Token Format

Bytes	Description
0	X'01' (flag indicating this is an internal key token)
1–3	Implementation-dependent bytes (X'000000' for ICSF)
4	Key token version number (X'00' or X'01')
5	Reserved (X'00')
6	Flag byte **Bit** **Meaning When Set On** **0** Encrypted key and master key verification pattern (MKVP) are present. This will be off for clear keys. **1** Control vector (CV) value in this token has been applied to the key. This will be off for clear keys. **2–7** reserved
7-15	Reserved (X'00')
16–23	A single-length key, the left half of a double-length key, or Part A of a triple-length key.
24–31	X'0000000000000000' if a single-length key, or the right half of a double-length operational key, or Part B of a triple-length operational key.
32–47	Reserved for clear key tokens (X'00’s')
48–55	X'0000000000000000' if a single-length key or double-length key, or Part C of a triple-length operational key.
56-58	Reserved (X'000000')
59 bits 0 and 1	B'00' reserved
59 bits 2 and 3	**B'00'** Indicates single-length key (version 0 only). **B'01'** Indicates double-length key (version 1 only). **B'10'** Indicates triple-length key (version 1 only).
59 bits 4 –7	B'0000'
60–63	Token validation value (TVV).