KEYXPART

KEYXPART is a sample ICSF utility that allows clear key parts to be entered into the ICSF CKDS as two key parts, thereby affording dual custody of these parts.

The application runs under a TSO ISPF session and communicates to the Key Custodians via ISPF panels.

This utility, like the Key Generation Utility Program (KGUP), does have clear key parts available in memory, and in some installations, this may be deemed unacceptable.
These areas are cleared by the application as soon as possible, but they do still exists. It also means that if connection to the TSO terminal is over an unsecured network connection, these key parts will be flowing in the clear from the terminal to the application. A locally coax attached terminal can minimize this exposure, but it is also the same exposure as exists with KGUP.

The program is invoked via the ISPF “6” (Issue TSO COMMAND) option from within ISPF “CALL ‘user.LOAD(KEYXPART)’ ”.

An initial screen prompts for a KEY LABEL (the name the key is to stored in the CKDS as) a key type (e.g. PINGEN, PINVER, MAC, IMPORTER etc.) and the 8, 16 or 24 bytes of the first key part. Each input field holds 16 hex digits representing each of the 4 bits of each key digit.
If the key label already exists, and it is desired to re-create the key, the KEY TYPE field may be filled with DELETE rather than the above PINGEN etc. examples. An existing key is never automatically overwritten. Now, also KEYGENKY or DUKPT derivation keys may be entered.

When ENTER is pressed, a second input screen is displayed. The desired KEY NAME and TYPE are carried over from the first panel. The second (and therefore last) key part can now also be entered.
Key part 2 must consist of the same number of key components (8, 16 or 24 bytes) as were entered for key part 1.
Each of these components is 8 bytes (16 hexadecimal digits). Parity for the combined key parts can also be forced to ODD.

If a key type is entered that is required by ICSF to be 16 bytes (32 digits), but only 8 bytes (16 digits) have been entered, then the program automatically replicates the first 8 bytes (left key half) into the second 8 bytes (right key half). This effectively creates a double length key (required by ICSF) that functions cryptographically like a single length key.
An example is a PIN encryption key (or KPE/IPINENC/OPINENC). Most devices still use a single length PIN encryption key, but ICSF requires that to be a double length key. The key custodian has the option of either entering only the 8 byte value or entering the same 8 byte value into the first two key entry fields.

DATA type keys (used for encryption/decryption) can be either single, double or triple length, therefore the third key entry filed can only be used for a DATA key type.

Once key custodian 2 has pressed enter, the program calculates a Key Check Value (clear key used

to encrypt 8 bytes hex zero) for the first key part, the second key part and the combined key part.
If these values match KCV's that had been given for the key parts, pressing ENTER completes the key function and at this point actually writes the new key into the CKDS.
At any point, ESC or PF3 can be pressed to terminate a function.
WHAT IT WILL NOT DO:
KEYXPART can not be used to enter EXPORTER/IMPORTER key types with the NOCV option.
In order to do this, the application would have to run AUTHORIZED (in MVS terms).
NOCV keys can still be entered via KGUP.
Existing key entries will NOT be overwritten. Existing entries can be deleted using the initial entry panel, entering the desired key label, and specifying DELETE for the key type.

INSTALLATION:
The application consists of an ASM (BAL) source module that must be compiled and link edited.
In the compile step, access to SYS1.MACLIB (default) and SYS1.MODGEN must be made available under the SYSLIB DD statement in the JCL.
The link edit (or BIND) step requires access to SYS1.ICSF.SCSFMOD0 and SYS1.ISPF.SISPLOAD in its SYSLIB DD statement.

Five ISPF panels must be placed into a panel library (LRECL 80).
These are EHNPAN01, EHNPAN02, EHNPAN03, EHNPAN04 and EHNPAN05.
A message file (EHN00) must be placed into a message library (LRECL 80).

If the panel and message library are not already accessible to the user's TSO ID, they will need to be concatenated before invoking KEYXPART.
I use a CLIST:

```
CONCATD DA('USER.PANELS') FI(ISPPLIB) AFTER
CONCATD DA('USER.MSG') FI(ISPMLIB) AFTER
```

Before beginning my ISPF "6" session.

CUSTOMIZATION:
Feel free to change the formats of the EHNPANxx panels as well as any colours used.
Just ensure that all variable fields still exist.

KEYXPART

Message file (EHN00):

```
EHN001E 'INVALID ENTRY’                    .ALARM=YES
'FIELD IS REQUIRED’

EHN002E 'INVALID INITIAL KEY'              .ALARM=YES
'INITIAL KEY NOT SPECIFIED OR LESS THAN 16 HEX CHARACTERS'

EHN003E 'INVALID KEY TYPE'                 .ALARM=YES
'KEY TYPE IS INVALID'

EHN004E 'KEY LENGTH MISMATCH'              .ALARM=YES
'KEY PART 1 AND 2 LENGTH MISMATCH'

EHN005E 'INVALID KEY LENGTH'               .ALARM=YES
'KEY TYPE AND LENGTH MISMATCH'

EHN006E 'CKDS ERROR'                       .ALARM=YES
'COULD NOT CREATE RECORD'

EHN007E 'ENCRYPTION FAILED'                .ALARM=YES
'ENCRYPTION OF THE KEY FAILED'

EHN008E 'CKDS ERROR'                       .ALARM=YES
'WRITE TO THE CKDS FAILED'

EHN009E 'CKDS ERROR'                       .ALARM=YES
'RECORD DELETE FAILED'
```