SIEMENS EDA

Calibre® TVFencrypt User's Manual

Software Version 2021.2
Document Revision 14

SIEMENS

Revision History

Revision	Changes	Status/Date
14	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released April 2021
13	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released January 2021
12	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released October 2020
11	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Calibre Release Notes* for this products are reflected in this document. Approved by Michael Buehler.	Released July 2020

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Siemens EDA documentation source. For specific topic authors, contact the Siemens Digital Industries Software documentation department.

Revision History: Released documents maintain a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation which are available on https://support.sw.siemens.com/.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction to Calibre TVFencrypt

Calibre TVFencrypt is a utility that encrypts Tcl Verification Format (TVF) files.

Calibre TVFencrypt Overview

Calibre TVFencrypt is a utility you use to safeguard, through encryption, sensitive data contained within TVF files. The **caltvfencrypt** utility creates encrypted TVF files that you can distribute to third parties without disclosing proprietary or other sensitive information.

If you distribute an encrypted TVF file, it is your responsibility to fully support it. Siemens Digital Industries Software can only support files that do not contain encrypted information.

_____ **Note** _____
In order to protect the proprietary and other sensitive information of our customers, Siemens Digital Industries Software cannot decrypt an encrypted file. Any customer who encrypts a file is solely responsible for maintaining its corresponding source information.

Workflow

The Calibre TVFencrypt utility (**caltvfencrypt**) reads a plain text TVF rule file and outputs an encrypted rule file. You can input the encrypted TVF rule file directly into Calibre or you can source the encrypted file from another plain text TVF file. When you input an encrypted TVF rule file into Calibre, different Tcl interpreters are created that perform different functions related to interpreting the TVF file.

Figure 1-1 illustrates a basic workflow of creating an encrypted TVF file from a plain text TVF file. The encrypted file is then input directly to Calibre or is sourced from a plain text TVF file after first being read by the Tcl interpreters.

Figure 1-1. Calibre TVFencrypt Workflow
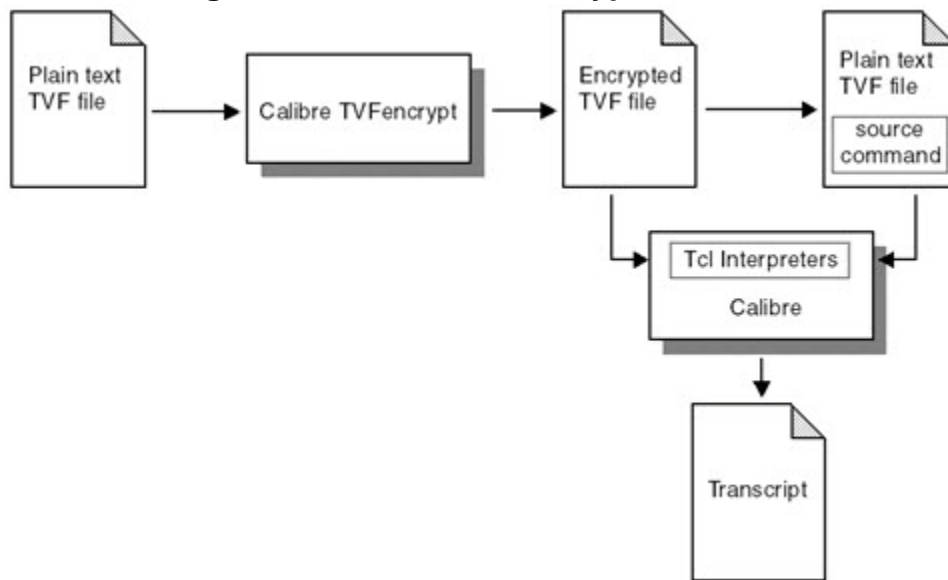
The process for creating and using an encrypted TVF file involves the following key steps:

- Preparing TVF Files for Encryption — Ensure Tcl built-in commands are used correctly, use TVF commands to apply global variables as needed, add emulated encryption statements, and set up sourcing of your encrypted TVF files.

- Creating an Encrypted TVF File — Use the appropriate **caltvfencrypt** command to encrypt your TVF files. You can encrypt a TVF file using one of the commands described in "Command Reference" on page 20.

- Using an Encrypted TVF File — Invoke Calibre and specify the encrypted TVF rule file or specify a TVF rule file that sources one or more encrypted rule files. Depending on how your encrypted TVF files are set up, the appropriate Tcl interpreters are created to interpret the TVF rule file(s).

License Requirements

Calibre TVFencrypt is available on all platforms supported by the Calibre product line.

The encrypting Tcl compiler (the TVFencrypt executable) requires the same license as SVRFencrypt. To run the compiler, you must set either the LM_LICENSE_FILE or MGLS_LICENSE_FILE environment variables to point to a valid license file or license daemon that contains a Calibre SVRFencrypt license feature.

For more information on supported platforms and licensing information, refer to the *Calibre Administrator's Guide*.

Syntax Conventions

The command descriptions use font properties and several meta-characters to document the command syntax.

Table 1-1. Syntax Conventions

Convention	Description
Bold	Bold fonts indicate a required item.
Italic	Italic fonts indicate a user-supplied argument.
`Monospace`	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
<u>Underline</u>	Underlining indicates either the default argument or the default value of an argument.
UPPercase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[ ]	Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.
' '	Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.
| or ||	Vertical bars indicate a choice between items. Do not include the bars when entering the command.
…	Three dots (an ellipsis) follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.

Example:
   DEVice {*element_name* ['('*model_name*')']}
    device_layer {*pin_layer* ['('*pin_name*')'] …}
   ['<'*auxiliary_layer*'>' …]
   ['('*swap_list*')' …]
   [<u>BY NET</u> | BY SHAPE]	

Prepare a TVF file for encryption by verifying support for Tcl built-in commands, defining global variables, and determining how the TVF files will be interpreted when running a Calibre job.

Tcl Built-in Command Support

There are some workarounds and requirements for using some of the Tcl built-in commands with encrypted files.

Info Built-in Command

The Tcl built-in **info** command cannot be used with encrypted files. One workaround is to use global variables (refer to "Global Variables" on page 17) to register and access procedures. In the following example, my_proc is called if it exists in *B.tbc*.

A.tvf (encrypted to A.tbc):

```
#!tvf
namespace import tvf::*

source B.tbc

if {[get_global_variable registered_procs("my_proc")] == 1} {
   my_proc
}
```

B.tvf (encrypted to B.tbc):

```
#!tvf
namespace import tvf::*

set global_variable registered_procs("my_proc") 1
proc my_proc {} {
   puts "contents of my_proc"
}
```

Package Provide and Package Require Built-in Commands

The **package require** command works with TVFencrypt as long as the Tcl file that contains the **package provide** command is not encrypted. For example, if the Tcl file *pkgIndex.tcl* includes:

```
package if needed etvf1 1.0 [list source [file join $dir p1.tcl]]
```

and the unencrypted Tcl file *p1.tcl* includes:

```
package provide etvf1 1.0
source dir1/e1.tbc
source dir1/e3.tbc
```

where the *.tbc* files are encrypted, then using this statement in a TVF rule file gives the expected results:

```
package require etvf1 1.0
```

Global Variables

There are some TVF commands you can use to define global variables that can be shared across the different Tcl interpreters (both encrypted and unencrypted).

The TVF commands for defining global variables are listed in Table 2-1. Each command name in the table links to the reference page in the *Standard Verification and Rule Format (SVRF) Manual* that provides information about the command.

Table 2-1. TVF Commands for Defining Global Variables

TVF Command	Summary
tvf::exists_global_variable	Checks for the existence of a global variable given its name and optional index values. Returns 1 if the variable exists, 0 otherwise.
tvf::get_global_variable	Retrieves the value of a TVF global variable given its name and optional index values. Returns the value of the global variable if it exists, and "" otherwise.
tvf::set_global_variable	Sets the value of a TVF global variable given its name and optional index values. The variable is created if it does not exist. Returns 1 on success, 0 otherwise.
tvf::unset_global_variable	Clears the global variable given its name and optional index values. Returns 1 if the global variable existed prior to calling this function, and 0 otherwise.

The argc, argv and argv0 global namespace Tcl variables can be accessed across interpreters (both encrypted and unencrypted). Table 2-2 provides a brief description of these variables along with the predefined values they are set to.

Table 2-2. Global Namespace Tcl Variables

Variable	Description	Predefined Value
argc	The number of list elements in argv.	0
argv	A list of command line arguments.	{ }
argv0	The name of the file that was executed.	Name of the rule file.

TVF File Configuration

The process you use to create an encrypted TVF file and then source the file determines the Tcl interpreter that is used when running a Calibre job.

If you want to:

- Use the user interpreter to interpret the encrypted TVF file:

  Create the encrypted TVF file using the caltvfencrypt (Unsecured Mode) command. Invoke Calibre and specify the name of the encrypted TVF file. Both the encrypted and unencrypted TVF files are read by the same user interpreter.

- Use one encrypted interpreter to interpret the encrypted TVF file:

  Create the encrypted TVF file using the caltvfencrypt command. Invoke Calibre and specify the name of the encrypted TVF file for the input rule file. The encrypted TVF rule file is read by an encrypted interpreter.

- Use separate encrypted interpreters for interpreting multiple encrypted TVF files:

  Create the encrypted TVF files using the caltvfencrypt command and specify different *id_string* values for the **-id** argument. Create an unencrypted TVF file that sources the encrypted TVF files. Invoke Calibre and specify the name of the unencrypted TVF file. The unencrypted TVF file is read by the user interpreter, while the encrypted TVF files are read by separate encrypted interpreters.

- Use the same encrypted interpreter to interpret multiple encrypted TVF files:

  Create the encrypted TVF files using the caltvfencrypt command and specify the same *id_string* value for the **-id** argument. Create an unencrypted TVF file that sources the encrypted TVF files. Invoke Calibre and specify the name of the unencrypted TVF file. The unencrypted TVF file is read by the user interpreter, while encrypted TVF files that have the same id are read by the same encrypted interpreter.

Related Topics

caltvfencrypt

caltvfencrypt (Unsecured Mode)

Chapter 3
Creating an Encrypted TVF File

Once you have prepared your TVF files for encryption, the process of creating an encrypted TVF file involves executing the **caltvfencrypt** command which invokes a Tcl compiler to encrypt the file.

Command Reference

There are two modes of the **caltvfencrypt** command: one mode creates an encrypted TVF file that runs on an encrypted interpreter while the other mode creates an encrypted TVF file that runs on the user interpreter (referred to unsecured mode). Typically, you will want to create an encrypted TVF file that runs on an encrypted interpreter.

caltvfencrypt

Creates an encrypted TVF file that runs on an encrypted interpreter when the file is used for a Calibre job.

Syntax

caltvfencrypt -id *id_string* [-new | -old] {**-stdin** | *input_file*} [*output_file*]
[-E [-tvfarg *tvf_arg_value*]] -debug

Parameters

- **-id** *id_string*

  A required argument that assigns a user-supplied *id_string* to the output file. Each output file identified with the same *id_string* is read into the same encrypted Tcl interpreter. If *id_string* contains spaces, it must be enclosed in quotes.

  > **Caution**
  >
  > Do not share or distribute the value of *id_string*. Using this string elsewhere may allow other TCL code to run in the same interpreter with the encrypted code.

  See also tvf::run_as_encrypted in the *Standard Verification Rule Format (SVRF) Manual*.

- -new | -old

  Optional encoding format switches that control encryption compatibility with older versions of Calibre. The switches work as follows:

  -new — Provides the new encoding format. TVF files encoded with this switch will not run on versions of Calibre earlier than 2014.4.

  -old — Provides the old encoding format. TFV files encoded with this switch will not run on future versions of Calibre that may use newer versions of the Tcl programming language.

If a switch is not specified, caltvfencrypt defaults to a dual encoding scheme for encrypting the TVF code.

The dual encoding scheme provides the following formats:

  o A new format provides forward compatibility for future versions of Calibre. Future versions may use newer versions of the Tcl programming language.

  o The original encoding format that provides back compatible operation in the older versions of Calibre.

Using the default dual encoding scheme results in a larger file size than previous releases. If the file size is a concern, then use one of the switches to control the output size. This will affect the compatibility of the encrypted file with either newer or older versions of Calibre.

> **Note**
>
> Default dual encoding is recommended for most situations. The TVF interface is subject to possible changes in syntax and compatibility in a release. An effort is made to maintain backward compatibility in the TVF interface. However, any changes can cause previously working TVF code to require maintenance in order to run on newer Calibre releases. The dual encoding encryption format is critical for maintaining the ability to repair and regenerate encrypted TVF files for future releases.

- **-stdin** | *input_file*

  A required argument set that specifies that the input to the compiler is read either from standard input (for example, from a pipe in a Tcl script), or from a user-supplied *input_file*. You must specify a value for *output_file* when using **-stdin**.

- *output_file*

  An optional argument that specifies a user-supplied output file. This argument is required if you specify **-stdin**.

  If you do not specify this argument, the output is written to a file created by replacing the file extension of *input_file* with ".tbc". For example, if the input file is "*test.tcl*", the output file becomes "*test.tbc*".

- -E [-tvfarg *tvf_arg_value*]

  An optional argument that checks for issues with an encrypted TVF file.

  Use the -E option with a compiled TVF file that generates one SVRF rule file. Rule files generated with parameter variations will not work with this option.

  The -tvfarg and *tvf_arg_value* parameter set specifies a user-defined value to pass to the TVF script, where *tvf_arg_value* is a valid string with no spaces. Use the tvf::get_tvf_arg command to access the value of *tvf_arg_value*.

- -debug

  An optional argument that directs Calibre to emit more debug information when executing an encrypted TVF file.

  By default, errors that occur in the encrypted interpreter produce very little information. The -debug argument allows more information about missing variables, call stacks, and so forth to be produced from the TVF environment. This information is helpful in debugging differences between unencrypted TVF runs and encrypted TVF runs.

Description

Invokes the Tcl compiler and creates an encrypted file from standard input or a user-supplied input file. The file runs on an encrypted interpreter when it is used for a Calibre job. Refer to "Tcl Interpreters and TVF Files" on page 29 for information on Tcl interpreters.

Examples

Example 1

The following example reads *width_check.tvf* from standard input and writes the encrypted output to *width_check.tbc* with the id check1.

```
cat width_check.tvf | caltvfencrypt -id check1 -stdin width_check.tbc
```

Example 2

This example creates three encrypted files named *rule1.tbc*, *rule2.tbc*, and *rule3.tbc*.

```
caltvfencrypt -id str1 rule1.tvf
caltvfencrypt -id str2 rule2.tvf
caltvfencrypt -id str1 rule3.tvf
```

An unencrypted TVF file contains the following **source** commands.

```
source rule1.tbc
source rule2.tbc
source rule3.tbc
```

Specifying the unencrypted TVF file as the rule file for a Calibre job creates four Tcl interpreters. Specifically, a master interpreter (m_interp), a user interpreter (u_interp, which reads the TVF rule file), and two encrypted Tcl interpreters. Because *rule1.tbc* and *rule3.tbc* have the same id string, they run on the same encrypted interpreter, while *rule2.tbc* runs on a separate encrypted interpreter.

caltvfencrypt (Unsecured Mode)

Creates an encrypted TVF file that is read by the user interpreter when the file is used for a Calibre job.

Syntax

caltvfencrypt -global_interp {**-stdin** | *input_file*} *output_file*

Parameters

- **-global_interp**

  A required argument that specifies to create an encrypted TVF file that will be read by the user interpreter when using the file for a Calibre job. An encrypted TVF file, when read by the user interpreter, is in *unsecured mode* allowing it to be accessed by other programs. The SVRF statements generated from the encrypted TVF file remain encrypted.

- **-stdin** | *input_file*

  A required argument that specifies the input for the Tcl compiler. Options include:

  - **-stdin** — Specifies the input is read from standard input (for example, from a pipe in a Tcl script).

  - *input_file* — Specifies the name of the TVF file to use as input to the Tcl compiler.

- *output_file*

  Specifies the name of an output file in which to write the encrypted TVF file. If you do not specify this parameter, the output is written to a file created by appending the ".tbc" suffix to the name of the specified *input_file*. For example, if the input file is *test.tcl*, the output file becomes *test.tbc*.

Description

Uses the same user (unencrypted) interpreter for running both the encrypted and unencrypted TVF files. Reading an encrypted file by a user interpreter is not recommended as, even though the file is encrypted, it can be accessed by other programs.

Examples

This example uses the user interpreter when encrypting the file named *drc.rules*. When the resulting encrypted file, *drc.tbc*, is used as input for a Calibre job the encrypted file is read by the user interpreter.

```
caltvfencrypt -global_interp drc.rules
```

TVF File Examples

Examples of a TVF file in its unencrypted and encrypted form illustrate the encryption capabilities of the **caltvfencrypt** utility.

Unencrypted TVF File

Below is an example of an unencrypted TVF file named *width_check.tvf*.

The encrypted version of this file is available in "Encrypted TVF File".

```
#!tvf

tvf::VERBATIM {

    layout path layout.gds
    layout system gds2
    layout primary '*'
    drc results database '/dev/null'

    layer metal 0

    METAL_WIDTH {
    // Metal width check. Metal width must be greater than or equal to
    // 3 microns except where metal length exceeds 5 microns; in that case,
    // metal width must be greater than or equal to 4 microns.

    long_metal = metal LENGTH > 5

    INTERNAL long_metal < 4

    short_metal = metal NOT LENGTH > 5

    INTERNAL short_metal < 3
    }
}
```

Encrypted TVF File

Below is an example of the encrypted TVF file named *width_check.tbc*.

The unencrypted version of this file is available in "Unencrypted TVF File".

```
#!tvf
if {[catch {package require calibre_enc 1.6} err] != 1} {
if {[catch {package require calibre_dec 1.0} err] == 1} {
    error "CALENC Loader while trying to load encrypted Tcl file: $err"
}
calibre_dec::calibre_enc_load_dec {
CALENC 3
%@4~G?^B8,+&Y,A>B&&:+&>T^NBA-?GHXTZ9,OEJ4CA]#]%!#9D&F?5\*DX*Y.5)^$<B!)A!!"
E,,RV&R2UT/\
   1W74`42FBQ!!"0=__073L+FB3K8YX"Z.MY:3P_QE[P:@_NU(?=.X4G5J0)3*&J8IVCZ>1.LO[.&MQ_!
   [I:^5AHBI5Z#D>.*!^*[6\A>WL]GU/B=;KE98R$Q_Y133F"/:)<$1_E>QE-69D.00IXT"\
   R:I^*8"ASL04W!YFT)68./F@K[(\BF2CM^$%'^";+LZ+DA0H3YA*JSYU!
G-T\,7HKNVC(4XN+1`PM=Q!!""0NHY=:X$.0D06P8X(0Y-BI?F^05H&B1?62(FL#SK7C:C\/C@[\
   (196:LW]C;7>94P5A$B)^L;4P:4'52Z.`3-08M$^00K=FR>1J.<<&+Q&P/
   ?(]KV0QS1YO83&H"K#4XV0.+OAD;*<%N,@:,N'5*%/5&B>%R:HUFLM/'?W]=$8_;[WBSDK-
   A%RS%U_.#GY0!
(LJE,+;1N8KT7:7G^):M9Q!!"FW(%??WE?B83"."/5I3&5)`B=)-L);,1M'J#Z2&S\MXHMCC#\
   &&;`G`H.`3UX]>DW#\;>1(@M?AUK<A9$*()BMC$L\#IZF.;!YC5&<Z707"I0,,_+9B$P]B]U,/
   #7>PSY:1+X-N7=7WOR!`2#I##/7=8C0+=J;VZ#^+-UZ?NJ4F(NE4T0X;Q\%O:C,1\F;TA!
HSB:\V:%*KF<R=@)5K1G&A!!"+Q1-CAA$(WVC(`^EN6PA3:2I_MS0U;Y$G/,%S="C)@RF/
   B?I8O[$=#U6;Y%#@BW-K,8JE"[A:>0E!?7/L65XJ8?[T.%OWYO;'H"V4_AE3KKN*[?AY+$Z.Z\
   Y!!0L+0%!":+$3N_C2YH)9:=L8.FFC$TB:L'&,=&P/BT^7Y:-2BI[JJ:1-W41?:0TZ%L)(OIV!
P9M08H.B&6++?/W$1N<WDA!!"[)2D.?)G_2OC#KRIGCE@HPPBX\
   1QD>MZ4JZ42)]]DCKLPBNSY;ZX_6`>(!D8J&M5DAR]@"E49U\][U<)O&J/
   KGCM*5]>NK<TG>%Q]@VQ&>`5X4['*4U6:>R^J7$]=;_Y!
}
} else {
if {[catch {package require calibre_enc 1.4} err] == 1} {
    error "CALENC Loader error: $err"
}
if {[catch {package require calibre_dc 1.0} err] == 1} {
    error "CALENC Loader while trying to load encrypted Tcl file: $err"
}
calibre_dc::calibre_enc_load_dc {
CALENC 2 0 1.4 8.4
%/9~8!0/,_'VU=S,X8E-LBC*Z.$7-4B'X2A=4,1C]!*\&6]!#&0_B*[+3JG:?C]X&YGRB1Q!!"
;M/,4:)+BS`W@SG!!NVX>1!!"A"[TME6D?$S6;0\P&>=WMT1X\7*W,8&O;4HDXTT-
   =M!YY%10&G4L['GKK\
   +*0*O[3;LDTAFW'2]N>.C_``FWP5;1@2Q[,MW7AYC6O7R7K+#!D%U#%4@D,E&BT%))!+'R0A-
   !B3+:^2OSS_NTHJKB1.E#*_RB\9!/>!5<&81=#VGKL/=)6$SSP<B]V\=+J\%(!
#18;]LX#%X[<[():MT+/HUA!!"`):>N%PX.1/IE1D\K]@FN:H#C<3W.(.<(N9\QN<8U0-RYS"6=Q\
   )R"T5.,_4#_N(7AY:*B]"</[@Q14^JT<'IW*B$,DF#$^Q_OS)52EG/>*D/
   72['=7`O)U$3)=DJ3Q0=5O;Q>\+UGV6YF^52`?D)>[2!I*;\X'U4X$TZIDG?]SR%GF/L=/
   JO_`]*EOE=:"F!
G&C`Z=Q?C[6']OI![H^A11!!"8$I=:\K()8HX;!5:9R9'?9*F_K%#-
   GMDK$*O=BZBC52MT%^;TNL64LF9/@9#X>Y[WV,53BJZ=&JH28XM?.K?'JTUNH0/2OL40%VC>H!MXGJ-
   ,5EP28Y@;-5;4H@_%@@TPBK_/S0UC:2N8_BO?5.S*2X4];M+[I(:0G`F9M\
   [JJ;P;^;'FS+`*74ES[^Y.1\7!
5A44]M/F`/,_J&BD$=_2?1!!"L)7HWGGJ))<(BR.3,3HM=&\
   ?]6^L.)EY:YE^89+1A6P$.?([^L7*Z><[&=P%,'PHZG#(B<?[/*3$Q#+PINBDC%YVO^&MR$.E,3\
   ;;K-F/Y-M0KK.?5PQQ(SH^J90*O<DG79I$R.<P.E6O$1.\
   ]HSO)SVI05>*`CXJK`KBJGZU(,:5T&W.:EJXD/*L6T!R5MI!
0'F;C$*P4<1'KRE<"]_X;A!!"%H8!&6(RW3KI(;"`'/^6>O/RS-92OG[I$)Z0^^JW3H,#.0AQI%/
   R]AZSM)?SJ>9H<_#:J+:B0(HDY3HB]11K([LXV!WZ-?-0-7ZB[3]&'\VQ;J<Y7%8<]N)31WKB4_-
   S@';LF][C79:(NK!",C`&ES!Y(2!()KB=<(W&#P"3I39'5>Q5550.LVZBRFR\/HL!!
(VOH7.XGQ9#/
   WC"[A=.JX1!!"]"]R2E7>G&Z_,V^RRL^$;;::+HK&0R:35?[UK<]#3Y?G!6H;DHI>SP(L:`<VX/-
   UKCYHP^33:_VY>*\T\,,#M[!!!"
```

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
}
}

#!tvf

if {[catch {package require calibre_enc 1.4} err] == 1} {
    error "CALENC Loader error: $err"
}
if {[catch {package require calibre_dc 1.0} err] == 1} {
    error "CALENC Loader while trying to load encrypted Tcl file: $err"
}
calibre_dc::calibre_enc_load_dc {
CALENC 2 0 1.4 8.4
%'1~F*;/8U6:4SL=#'%"J7FT3^:LIT\%Y)8WS0DT6__"F[A!#"^FZ!*3:M;V'^E*S]-
XWXQ!!"
MEH24B&3IFU]2'5J$J5#Z!!!"X<LG!S(X$/3X'E(5,,XJ"JG*^!)@_AKW3A4L-LRYUB$6+GO/
X;VA][9)&>>_#4/:U;-R0MJ;D^J&7L_1ZGA?(9;K)WR5-8>Y^D0^1(>$079T['*A$Q;)S/
ZH.=<5\]9/)6QPB`[+8_>,OI.U7+34R-BIE;8'&?#*2;]KT41H!3.*OX-_ZK\9S^]'J-
X1TRZL:!F8.>QJ;EOCA5`=:2*#+3XI)S+B)3T]WK@B:D^D\\RL:47A1.G85'QCFR]\#ZY,:!&
60?TNAJ)/#W0V-W$,0DRK<?==BX-Z)P0$W):%<87P6;<-B=6=MZR2;
-J92\L>],TOVV^CM60I`2,P^)0BJ'5UP3'PKMZ$986/
E'AB@WKI.*TJ3O@%AH%LO&;["57<M6!V`<G00)=K9J,_=\8MR*K2O^A1:%'W2B:ME\JPZN#L7
LLJL,OH/0!UB'J-Z200T820Z""FI7L:KKV]]Y=]'S@?Z15J`8H?R@*<1.HE`]6RIS9)1/
;.*384]M@HVR[^>;_<V=K!67:HI(?^^!D3O[IRT8Y&!BQ82ZLYIXXNV*."S4Z$4_@.V<#!9%!
>1!X?N+$5_O5K

...

YF]]O.0$5@V#HHU=+:=*6_Y11*(N^)Z@SCR'0L/1L7694%@KA*;738?VX'8YO<JL/
'3E!FZ\"!_"F9D>(R`P5%(?H@1B>:@CK-
F?(_1&KWS$$C]1KSI_KKQVGCF?X1!O@<$85W77B_AD#CI`"`G^0Y.FRXZ'DT!6@LZ!?3:;?A"
)5/~'EGE==ON/:(3`T]45`H)^.,O0VX?'UPIM(&C!?XF-
'H!T'HZ=8<OB68N?U2HIY'8!$&S[X*YU,EE=1FG#N^ZN1D3";=-!2PPJ=V1:X["(I:-
<?O'2O^[\WWK)^7J9!%[NOX*=SJ5+IK/
8V]`H@8F6VVX5B51225O%5SEM^DJ9SL<!KC!JQ@RJ@UC/.@NW-
$WSPQ5RA%JY)[ZR1"UBVT!#GI["]'1:T-RN)-
'+U8I`^3'&)PX4\Q.0=^X:Q"B02PV#KV7XH!S(E[_"R>!$C#U.%S!LB:9#N?3(W("Z;"5C]V"
C*L40.^(ALR,PP[RD5?!;?AQ>[H)3+2ZTO"LC$TW)T'`41/
XGH*K`NJ(=8&5YP.'<S?=O1I>_QI.0QPSEC44D3]`)M\_5H!3BR=1&R$9(WRE$-
]:(TH7`%'LE=X0>S>-
@9>BK5M:A<\B(!;)C]BCP>IW202E,E<!6ZY3'4&7L`>YS<L3MZQ7D;TV)]C:I4_R@%Y+?S(J"
1_Z6`PL%*R5(/%B.^2I/+A/
N&X`&<%I'H5*&OGE7IH]<_QB(Z$G`&IE9>5"XCABJL9\37EV,%KHI$VO0K&()\LT+3QH#:#D-
'^C8LEC'&ARQ-ETO2X")[ZIGDU]BG3O>D`HZ&<=2R)+=QA7R>_<9&G=F4+(CMY!#_}
```

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

When Calibre reads a TVF rule file, at least two Tcl interpreters are created: a master interpreter and a user interpreter. If the rule file includes encrypted TVF code, then one or more encrypted interpreters are created depending on how the encrypted file is sourced.

Tcl Interpreters and TVF Files

The master interpreter (m_interp) reads Tcl code to define the TVF namespace. The user interpreter (u_interp) reads the TVF file specified as the Calibre rule file. If the rule file includes any encrypted files specified with the Tcl **source** command, then each encrypted file is read into an encrypted interpreter (e_interp).

Therefore, the interpreter running unencrypted TVF code is not the same interpreter running the encrypted TVF code. You can also source an encrypted file from within another encrypted TVF file as shown in Figure 4-1. If the encrypted TVF files are created using different *id_string* values, then separate encrypted interpreters are created for each encrypted TVF file.

Figure 4-1. Interpreter Structure
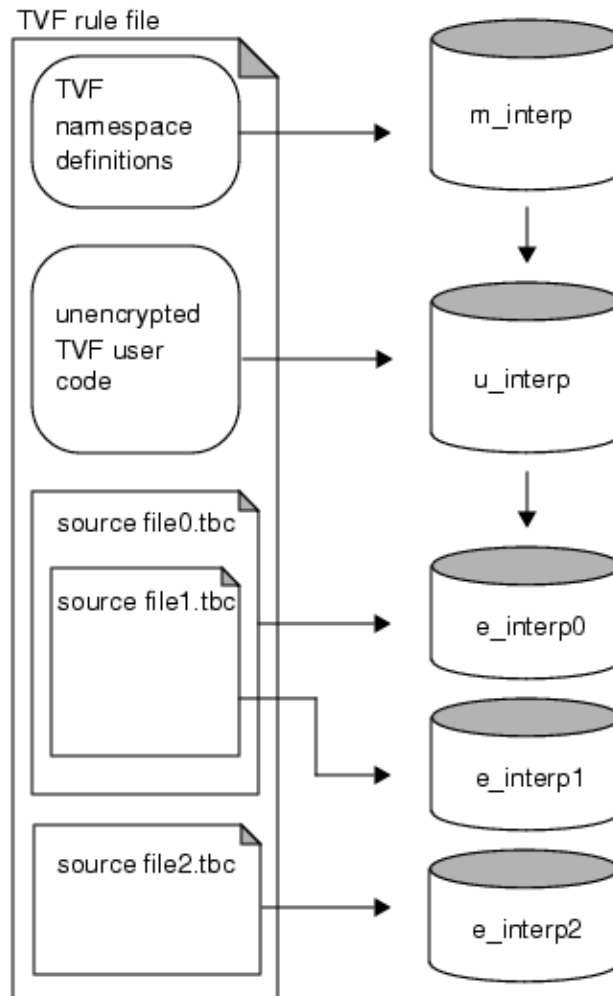
You can allow multiple encrypted TVF files to be read into the same encrypted Tcl interpreter by:

- Specifying the same ***id_string*** for each file when using the caltvfencrypt ***id_string*** argument, and

- Nesting calls to the files.

Encrypted files that do not meet both of these conditions are read into separate encrypted Tcl interpreters.

For example, in Figure 4-2, *file0.tbc* and *file1.tbc* are encrypted using the same ***id_string*** and are also nested. This allows the files to be read into the same encrypted Tcl interpreter. The encrypted file *file2.tbc* has a different ***id_string*** and is not nested, causing it to be read into a different encrypted Tcl interpreter.

Figure 4-2. Reading Multiple Files into an Encrypted Tcl Interpreter
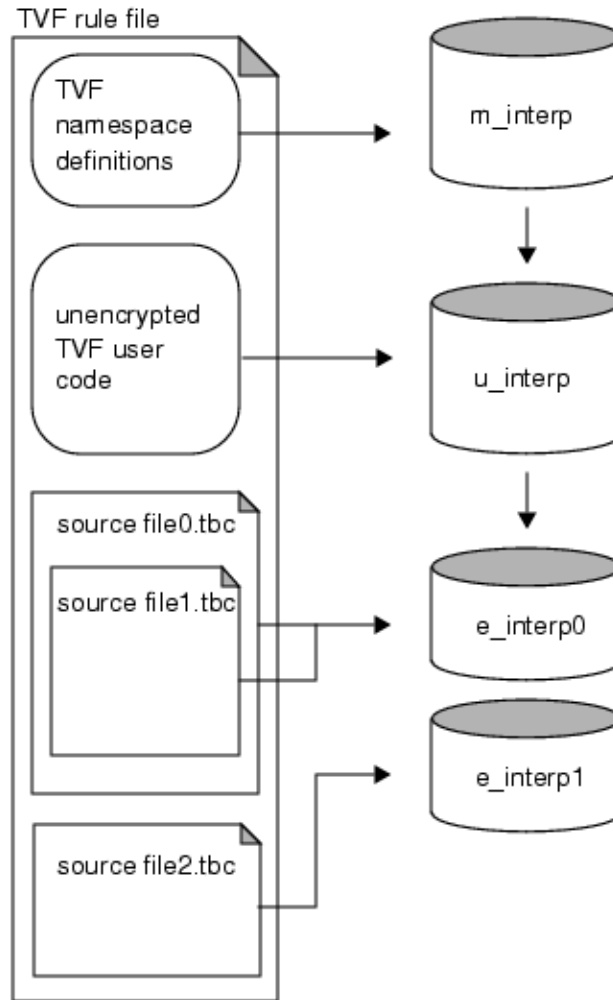
Global or namespace variables are not visible between interpreters, with the exception of global variables defined using certain commands (refer to "Global Variables" on page 17). Tcl procedures are generally not callable between interpreters, with two exceptions:

- Tcl procedures within encrypted Tcl interpreters are callable from the user interpreter, and from other encrypted Tcl interpreters.

- Procedures inside the TVF namespace of the master interpreter are callable from the user interpreter or any encrypted Tcl interpreter.

Namespace code cannot be used across interpreters. For example, if procedures within namespace ABC are defined in both *a.tvf* and *b.tvf*, the two files must be compiled with the same *id_string* when using the **caltvfencrypt -id** argument.

Transcript Output

To run an encrypted TVF file, you can either input to Calibre an unencrypted TVF file that contains a **source** command pointing to one or more encrypted files, or input an encrypted TVF file to Calibre directly.

In this example, the TVF file named *rules.tvf* contains the statement that sources the encrypted TVF file named *width_check.tbc* as shown here:

```
#!tvf

source width_check.tbc
```

The examples in this section compare the output from using the unencrypted *width_check.tvf* file and the encrypted *rules.tvf* file. The differences in output are highlighted.

Unencrypted Output

This example shows the transcript output from using the *width_check.tvf* file.

```
:-------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----             STANDARD VERIFICATION RULE FILE COMPILATION MODULE         -----
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------


--- RULE FILE = width_check.tvf

#!tvf

tvf::VERBATIM {

  layout path layout.gds
  layout system gds2
  layout primary '*'
  drc results database '/dev/null'

  layer metal 0

  METAL_WIDTH {
  // Metal width check. Metal width must be greater than or equal to
  // 3 microns except where metal length exceeds 5 microns; in that case,
  // metal width must be greater than or equal to 4 microns.

  long_metal = metal LENGTH > 5

  INTERNAL long_metal < 4

  short_metal = metal NOT LENGTH > 5

  INTERNAL short_metal < 3
  }

}
...

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----                      CALIBRE::DRC-F - EXECUTIVE MODULE                 -----
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

metal = OR metal
----------------
metal (TYP=1 CFG=1 ECT=14 OCT=11 SRT=1 CMP=F MPN=5)
CPU TIME = 0  REAL TIME = 0  LVHEAP = 0/0/1  OPS COMPLETE = 1 OF 5

Original Layer metal DELETED -- LVHEAP = 0/0/1

METAL_WIDTH::long_metal = LENGTH metal > 5
METAL_WIDTH::short_metal = NOT LENGTH metal > 5
-----------------------------------------------
METAL_WIDTH::long_metal (TYP=2 CFG=1 ECT=0 OCT=0 SRT=1 CMP=F MPN=0)
METAL_WIDTH::short_metal (TYP=2 CFG=1 ECT=20 OCT=20 SRT=1 CMP=F MPN=5)
CPU TIME = 0  REAL TIME = 0  LVHEAP = 0/0/1  OPS COMPLETE = 3 OF 5

Layer metal DELETED -- LVHEAP = 0/0/1
```

```
METAL_WIDTH::<1> = INT METAL_WIDTH::long_metal < 4
----------------------------------------------------
METAL_WIDTH::<1> (TYP=3 ECT=0 CCT=0)
CPU TIME = 0   REAL TIME = 0   LVHEAP = 0/0/1   OPS COMPLETE = 4 OF 5

Layer METAL_WIDTH::long_metal DELETED -- LVHEAP = 0/0/1

Layer METAL_WIDTH::<1> DELETED -- LVHEAP = 0/0/1

METAL_WIDTH::<2> = INT METAL_WIDTH::short_metal < 3
----------------------------------------------------
METAL_WIDTH::<2> (TYP=3 ECT=20 CCT=10)
CPU TIME = 0   REAL TIME = 0   LVHEAP = 0/0/1   OPS COMPLETE = 5 OF 5

Layer METAL_WIDTH::short_metal DELETED -- LVHEAP = 0/0/1
Layer METAL_WIDTH::<2> DELETED -- LVHEAP = 0/0/1

WRITE to ASCII DRC Results Database /dev/null COMPLETED

DRC RuleCheck METAL_WIDTH COMPLETED. Number of Results = 10

Cumulative ONE-LAYER BOOLEAN Time: CPU = 0   REAL = 0
Cumulative ONE-LAYER DRC Time: CPU = 0   REAL = 0
Cumulative MISCELLANEOUS Time: CPU = 0   REAL = 0
Cumulative RDB Time: CPU = 0   REAL = 0

--- CALIBRE::DRC-F EXECUTIVE MODULE COMPLETED.  CPU TIME = 0   REAL TIME = 0
--- TOTAL RULECHECKS EXECUTED = 1
--- TOTAL RESULTS GENERATED = 10
--- DRC RESULTS DATABASE FILE = /dev/null (ASCII)
...
```

Encrypted Output

This example shows the transcript output from using the *rules.tvf* file that sources the encrypted TVF file named *width_check.tbc*.

```
    --------------------------------------------------------------------------------
    --------------------------------------------------------------------------------
    -----             STANDARD VERIFICATION RULE FILE COMPILATION MODULE        -----
    --------------------------------------------------------------------------------
    --------------------------------------------------------------------------------


    --- RULE FILE = rules.tvf


    #!tvf


    source width_check.tbc


    ...
    --------------------------------------------------------------------------------
    --------------------------------------------------------------------------------
    -----                     CALIBRE::DRC-F - EXECUTIVE MODULE                  -----
    --------------------------------------------------------------------------------
    --------------------------------------------------------------------------------


    metal = OR metal
    ----------------
    metal (TYP=1 CFG=1 ECT=14 OCT=11 SRT=1 CMP=F MPN=5)
    CPU TIME = 0   REAL TIME = 0   LVHEAP = 0/0/1   OPS COMPLETE = 1 OF 5


    Original Layer metal DELETED -- LVHEAP = 0/0/1


    <ENCRYPTED OPERATION(S)>
    -----------------------
    <ENCRYPTED_LAYER_2> (HIER-FMF TYP=2 CFG=1 HGC=16 FGC=16 HEC=16 FEC=16 IGC=0 ...)
    <ENCRYPTED_LAYER_3> (HIER-FMF TYP=2 CFG=1 HGC=0 FGC=0 HEC=0 FEC=0 IGC=0  ...)
    CPU TIME = 0   REAL TIME = 0   LVHEAP = 1/3/4   OPS COMPLETE = 3 OF 5   ELAPSED TIME = 1


    Layer metal DELETED -- LVHEAP = 1/3/4


    <ENCRYPTED OPERATION(S)>
    -----------------------
    <ENCRYPTED_LAYER_4> (HIER TYP=3 HGC=0 FGC=0 HEC=0 FEC=0 IGC=0)
    CPU TIME = 0   REAL TIME = 0   LVHEAP = 0/0/1   OPS COMPLETE = 4 OF 5


    Layer <ENCRYPTED_LAYER_2> DELETED -- LVHEAP = 0/0/1


    Layer <ENCRYPTED_LAYER_4> DELETED -- LVHEAP = 0/0/1


    <ENCRYPTED OPERATION(S)>
    -----------------------
    <ENCRYPTED_LAYER_5> (TYP=3 ECT=20 CCT=10)
    CPU TIME = 0   REAL TIME = 0   LVHEAP = 0/0/1   OPS COMPLETE = 5 OF 5


    Layer <ENCRYPTED_LAYER_3> DELETED -- LVHEAP = 0/0/1


    Layer <ENCRYPTED_LAYER_5> DELETED -- LVHEAP = 0/0/1


    WRITE to ASCII DRC Results Database /dev/null COMPLETED


    DRC RuleCheck METAL_WIDTH COMPLETED. Number of Results = 10


    Cumulative ONE-LAYER BOOLEAN Time: CPU = 0   REAL = 0
    Cumulative ONE-LAYER DRC Time: CPU = 0   REAL = 0
```

```
Cumulative MISCELLANEOUS Time: CPU = 0   REAL = 0
Cumulative RDB Time: CPU = 0   REAL = 0

--- CALIBRE::DRC-F EXECUTIVE MODULE COMPLETED.   CPU TIME = 0   REAL TIME = 0
--- TOTAL RULECHECKS EXECUTED = 1
--- TOTAL RESULTS GENERATED = 10
--- DRC RESULTS DATABASE FILE = /dev/null (ASCII)
```

Appendix A
Emulated Encryption

You can use emulated encryption in cases where you do not want to encrypt a rule file, but you want to encrypt the output from a Calibre job. There are two directives, #PRAGMA ENCRYPTED and #PRAGMA ENDCRYPTED, that identify the start and end sections containing the SVRF operations you want to encrypt in the output.

PRAGMA Directives

All lines between the #PRAGMA ENCRYPTED and #PRAGMA ENDCRYPTED directives are treated by Calibre as if they were encrypted: they are not echoed into the transcript.

Instead, the string "<ENCRYPTED *>" is echoed as shown in the Emulated Encryption Example. The output of all SVRF operations contained between the two statements is suppressed in the same way as it would be done if the SVRF operations were encrypted. In other words, the results of executing Calibre with code between #PRAGMA ENCRYPTED and #PRAGMA ENDCRYPTED is the same as the result obtained with a rule file where the same code is actually encrypted. The rule file itself is not encrypted and contains all SVRF statements in plain text.

Nested emulated encryption statements are supported; everything between the outermost #PRAGMA ENCRYPTED and #PRAGMA ENDCRYPTED statements is protected. Emulated encryption statements should not be used in encrypted sections of SVRF files. It is valid to use both emulated and SVRF encryption in the same rule file as long as the sections affected by each do not overlap.

Emulated Encryption Example

In this example, you create a TVF file that contains #PRAGMA ENCRYPTED and #PRAGMA ENDCRYPTED statements. You then compare the output from running Calibre with and without these statements.

Procedure

1. Create an ASCII file named *A.tvf* and add the following:

```
#!tvf
namespace import tvf::*

source B.tvf

OR3 L1 L2 L4

LAYOUT PATH "lab1a.gds"
LAYOUT PRIMARY "TOP"
LAYOUT SYSTEM GDSII
DRC RESULTS DATABASE "lab1a.db" ASCII

LAYER L1 0
LAYER L2 1
LAYER L4 2
```

2. Save and close the file.

3. Create an ASCII file named *B.tvf* and add the following:

```
#!tvf
namespace import tvf::*

proc OR3 {L1 L2 L4} {
SETLAYER AA = L1 or L2
SETLAYER BB = AA or L4
RULECHECK AA {COPY AA}
RULECHECK BB {COPY BB}
}
```

4. Save and close the file.

5. Execute Calibre as follows and output the transcript to a file named *unencrypt.log*:

```
calibre -drc -hier A.tvf |tee unencrypt.log
```

   When you execute Calibre, the **source** command in *A.tvf* sources the file *B.tvf*.

6. Edit the file *B.tvf* and add the #PRAGMA ENCRYPTED and #PRAGMA ENDCRYPTED statements:

```
#!tvf
namespace import tvf::*

proc OR3 {L1 L2 L4} {
VERBATIM "#PRAGMA ENCRYPTED"
SETLAYER AA = L1 or L2
SETLAYER BB = AA or L4
RULECHECK AA {COPY AA}
RULECHECK BB {COPY BB}
VERBATIM "#PRAGMA ENDCRYPTED"
}
```

7. Save and close the file.

8. Execute Calibre as follows and output the transcript to a file named *encrypt.log*:

    ```
    calibre -drc -hier A.tvf |tee encrypt.log
    ```

9. Compare the *unencrypt.log* and *encrypt.log* log files you output in step 5 and 8.

Results

When the #PRAGMA statements are not present in *B.tvf*, the operations inside the OR3 procedure appear in the *unencrypt.log* transcript:

```
...
AA = L1 OR L2
-------------
AA (HIER TYP=1 CFG=1 HGC=9 FGC=50 HEC=72 FEC=400 IGC=16 VHC=F VPC=F)
CPU TIME = 0   REAL TIME = 0   LVHEAP = 1/3/4   OPS COMPLETE = 3 OF 5   ELAPSED
TIME = 0

Layer L1 DELETED -- LVHEAP = 1/3/4

Layer L2 DELETED -- LVHEAP = 1/3/4

DRC RuleCheck AA COMPLETED. Number of Results = 9 (50)
L3 = OR L3
----------
L3 (HIER TYP=1 CFG=1 HGC=70 FGC=472 HEC=280 FEC=1888 IGC=32 VHC=F VPC=F)
CPU TIME = 0   REAL TIME = 0   LVHEAP = 1/3/4   OPS COMPLETE = 4 OF 5   ELAPSED
TIME = 0

Original Layer L3 DELETED -- LVHEAP = 1/3/4

BB = AA OR L3
-------------
BB (HIER TYP=1 CFG=1 HGC=44 FGC=286 HEC=212 FEC=1344 IGC=0 VHC=F VPC=F)
CPU TIME = 0   REAL TIME = 0   LVHEAP = 1/3/4   OPS COMPLETE = 5 OF 5   ELAPSED
TIME = 0

Layer AA DELETED -- LVHEAP = 1/3/4

Layer L3 DELETED -- LVHEAP = 1/3/4

Layer BB DELETED -- LVHEAP = 1/3/4
...
```

When the #PRAGMA statements are present, the operations inside the OR3 procedure are replaced with the string "<ENCRYPTED …> in the *encrypt.log* transcript:

```
...
<ENCRYPTED OPERATION(S)>
-----------------------
<ENCRYPTED_LAYER_4> (HIER TYP=1 CFG=1 HGC=9 FGC=50 HEC=72 FEC=400 IGC=16
VHC=F VPC=F)
CPU TIME = 0   REAL TIME = 0   LVHEAP = 1/3/4   OPS COMPLETE = 3 OF 5   ELAPSED
TIME = 0

Layer L1 DELETED -- LVHEAP = 1/3/4

Layer L2 DELETED -- LVHEAP = 1/3/4

DRC RuleCheck AA COMPLETED. Number of Results = 9 (50)

L3 = OR L3
----------
L3 (HIER TYP=1 CFG=1 HGC=70 FGC=472 HEC=280 FEC=1888 IGC=32 VHC=F VPC=F)
CPU TIME = 0   REAL TIME = 0   LVHEAP = 1/3/4   OPS COMPLETE = 4 OF 5   ELAPSED
TIME = 0

Original Layer L3 DELETED -- LVHEAP = 1/3/4

<ENCRYPTED OPERATION(S)>
-----------------------
<ENCRYPTED_LAYER_7> (HIER TYP=1 CFG=1 HGC=44 FGC=286 HEC=212 FEC=1344
IGC=0 VHC=F VPC=F)
CPU TIME = 0   REAL TIME = 0   LVHEAP = 1/3/4   OPS COMPLETE = 5 OF 5   ELAPSED
TIME = 0

Layer <ENCRYPTED_LAYER_4> DELETED -- LVHEAP = 1/3/4

Layer L3 DELETED -- LVHEAP = 1/3/4

Layer <ENCRYPTED_LAYER_7> DELETED -- LVHEAP = 1/3/4
...
```

Index

Third-Party Information

Details on open source and third-party software that may be included with this product are available in the *<your_software_installation_location>/legal* directory.