Mentor®
A Siemens Business

Questa® SIM Encryption User's Manual

Including Support for ModelSim®

Software Version 2020.4

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The Questa SIM simulator encryption features allow Intellectual Property (IP) authors to deliver encrypted IP code for a wide range of EDA tools and design flows. This user guide describes the various commands and flows for supporting IP encryption.

Preparing for Encryption

The methodology for encryption depends on user needs and the required end result.

Basic Uses of Encryption

In general, there are two purposes for encryption that determine the encryption flow:

- *External IP protection*. For example, you have created an Ethernet controller which is meant for various outside customers and users. In this scenario you will be determining which set of trusted vendor tools can decrypt your IP. In turn this requires managing a set of public encryption keys. In addition, this would include defining what set of objects will be visible to users of the IP.

- *Internal single tool regression obfuscation.* In these scenarios, the IP remains in-house and is used for such purposes as black box testing. This scenario typically requires only one encryption key.

Cryptography Information

After choosing which third party tools to support, you will need a public encryption key for each one. This information may be kept in your network keyring library. You will need to list which versions of the IEEE encryption protocols they support. As a result of IEEE support, you must determine a common data method understood by all these tools.

Name /Object Visibility

You need a list of object names and interfaces from the target source HDL connections that need to be either disclosed or inherently visible to the end user. You will need to determine what method to use to release these names. For HDL access only, you can disclose names to the end user without making them strictly visible to the tools. For any other use, such as waveform logging, you will need to manage the visibility of these names at encryption. Text selection is either whole-file encryption or using the embedded pragmas.

IEEE Protocol IP Encryption Support

Mentor Graphics supports the following IEEE protocols for IP encryption:

- The Questa SIM simulator supports version 2 of the IEEE 1735-2014 standard for encryption interoperability. It addresses use models, algorithm choices, conventions, and minor corrections to the HDL standards to achieve useful interoperability.

  The IEEE Std 1735-2014 is a clarification of the separate Verilog and VHDL definitions of source protection and applies to both languages. It addresses the inter-operable (that is, digital envelope concept) parts incompletely defined for Verilog and VHDL. It also describes using standard algorithms to encrypt/encode the original source code into a form that any compliant tool can use.

- Currently, the Questa SIM simulator does not support rights management nor license management.

- The Questa SIM simulator supports VHDL, Verilog, and SystemVerilog IP code encryption by means of encryption envelopes.

- VHDL encryption is defined by the IEEE Std 1076-2008, section 24.1 (titled “Protect tool directives”) and Annex H, section H.3 (entitled “Digital envelopes”).

- Verilog encryption is defined by IEEE Std 1364-2005, section 28; and SystemVerilog encryption is defined by the IEEE Std 1800-2012, section 34 (both sections are entitled “Protected envelopes”).
- The digital envelopes usage model, as presented in Annex H section H.3 of these standards, is the recommended methodology for users of VHDL and Verilog. You should obtain these specifications for further reference.

The vencrypt and vhencrypt Encryption Commands

You encrypt Intellectual Property through the shell command line programs **vencrypt** and **vhencrypt**. The shell command **vencrypt** encrypts Verilog and SystemVerilog IP. The shell command **vhencrypt** encrypts VHDL IP.

After processing a clear text source file, both commands generate an encrypted file appending a “p” to the original file name. For example, the encrypted version of *flipflop.v* becomes *flipflop.vp*. The IEEE encryption standards also refer to the contents of the encrypted file as a *decryption envelope*. Targeted tools use decryption envelopes to decrypt and run the encrypted files.

Both commands provide a variety of arguments to control encryption. In addition, the *modelsim.ini* file has entries for controlling encryption similar to the command line arguments. Use the *modelsim.ini* file to set default encryption settings.

For a complete listing of all command line arguments and *modelsim.ini* entries, see the Reference Chapter.

Table 1-1 lists the encryption command arguments that support the IEEE 1735 protocol, but are not backward compatible with previous versions of these commands.

Table 1-1. Encryption Arguments Supporting IEEE 1735

Argument	Description
-common <commonfile>	Specifies the file of a common set of pragmas.
-keyring <directory>	Specifies the location of the *keyring* directory. The default is *<install>/keyring*
-toolblock <keyfile>,[<rightsfile>]	Specifies filename with a targeted tool’s public encryption key. Optionally, specifies file with rights attributes.

The shell commands do not encrypt files into a library, nor do they process macros or handle Verilog arguments. TheQuesta SIM **vhencrypt** command for VHDL works the same as the **vencrypt** command (though VHDL does not have macros). Both commands recognize the Questa SIM public encryption key. The commands use the Questa SIM public encryption key if you do not specify any other key. You can find the details of the public encryption key in the *./key* directory in the files MGC-VERIF-SIM-RSA-2 and MGC-VERIF-SIM-RSA-3.

The **vhencrypt** and the **vencrypt** commands encrypt marked off source areas using the Advanced Encryption Standard (AES) algorithm. The commands generate a random key called a "session key." The Questa Sim encyption commands supports AES128, AES192, and AES 256 bit keys. The default is AES128. This session key itself is encrypted and encoded using each public key found in the pragmas that form the encryption envelope.

You can specify the algorithm for encrypting the session key using the -data_method argument. The session key is generated into the **key block**, and that **key block** is encrypted using the "RSA" method. The default is AES128. The commands communicate the session key to the decrypting tool by means of a **KEY_BLOCK** in the decryption envelope.

The only supported asymmetric encryption method is RSA. This method is supported only for specifying key information, not for encrypting IP source code (that is, only for key methods, not for data methods).

All encryption algorithms produce byte streams that contain non-graphic characters. Therefore, there needs to be an encoding mechanism to transform the arbitrary byte streams into portable sequences of graphic characters. The supported encoding methods are:

- uuencode
- base64

Base 64 encoding is the default method and the recommended encoding for all applications.

Pragma Usage

You can encrypt your IP using a variety of IEEE defined pragmas.

Supported Pragmas

Questa SIM supports most IEEE 1735 version 2 pragmas.

Table 1-2. Supported Encryption Pragmas

Pragma keywords	Purpose
data_method	Specifies data block encryption algorithm.
key_keyname key_keyowner key_public_key key_method rights_digest_method	Defines attributes of a public encryption key.

begin
end
	Identifies the start and end of a text segment to be encrypted.

viewport interface_viewport	Specifies what objects are visible to a user of a supported tool. The use of viewports is a preferred alternative to using multiple encryption envelopes.
control license_proxyname license_public_key license_attributes license_public_key_method license_keyowner license_keyname license_symmetric_key_method	Optional specific conditional usage rights (common and per-tool).

Table 1-2. Supported Encryption Pragmas (cont.)

Pragma keywords	Purpose
version author author_info encoding (deprecated)	Optional pragmas. The version pragma is typically set by the encryption commands.

begin_toolblock
end_toolblock
begin_commonblock
end_commonblock
	Defines begin and end for common and tool blocks.

Some pragmas have set values and default values. Others have no defaults. The following table shows set and default values.

Table 1-3. Pragmas with Set and Default Values

Pragma	Values and Defaults
data_method	aes128-cbc | aes192-cbc | aes256-cbc Default: aes128-cbc
key_keyowner	If none specified, then defaults to "Mentor Graphics Corporation"
key_keyname	If none specified, then defaults to “MGC-VERIF-SIM-RSA-2”
key_method	If none specified, then defaults to “rsa”
rights_digest_method=	If none specified, then defaults to "sha256”
key_public_key	If none specified, then defaults to the Questa Sim public key.

encoding	Base64 is the only available IEEE 1735 version 2 encoding method.
	enctype = "base64"
	line_length = 64 ,
	bytes = 576

version	Specifies which IEEE 1735 version to use. Can be 0, 1, or 2. The encryption commands infer a version level based on the pragmas it encounters.

Unsupported Pragmas

Questa SIM does not support the following protection directives:

- Digest pragmas.
- All license pragmas.

Pragmas for Marking Clear Text Source for Encryption

The recommended methodology for creating encrypted source files is to move all encryption recipe information out of the source IP files, and only use the following pragmas to control visibility.

- Verilog
  - `pragma protect begin
  - `pragma protect end
  - `pragma protect viewport
  - `pragma interface_viewport
  - pragma protect
- VHDL
  - `protect begin
  - `protect end
  - `protect viewport
  - `protect interface_viewport

When the encryption commands encounter a contiguous block of targeted source code, the tools encrypt the marked text into a single decryption envelope. You may annotate the HDL to identify one or more such targeted text blocks, and the tools then encrypt each block into an equivalent decryption envelope. The encryption tools transfer source text verbatim, and in order they appear in the encrypted file.

If the encryption commands do not find begin and end protect directives, they will add them to the targeted files, and rescan the files for any protection or visibility directives (viewports).

Viewport pragmas disclose names from within the encrypted HDL source code, making them visible to downstream tools as if they were not encrypted. Viewports offer more efficient control of visibility than multiple envelopes. Visibility control pragmas are the only pragmas that are supplied in source text between the begin and end pragmas, and are therefore encrypted as part of the source text encryption.

Viewports are optional pragmas inserted at encryption. You may choose not to add any viewports.

Common Blocks

The common block is used to collate licensing and rights pragmas common to all tools.

It may be empty, in which case you specify an empty common block. The target tools infer that all rights are delegated. The IEEE Standard. 1735 specifies that a common block is optional. The following example shows rights delegation:

```
`protect begin_commonblock
`protect control error_handling = "delegated"
`protect control runtime_visibility = "delegated"
`protect control child_visibility = "delegated"
`protect control decryption = "delegated"
`protect end_commonblock
```

Comments in Source Files and Encryption Recipes

Recipe files are meant to be language-neutral, and therefore suitable for either SystemVerilog (with its `pragma protect syntax) and VHDL (with its `protect syntax).

The encryption tools allow comments, but must begin with either the "//" or "--" characters, independent of the targeted language.

Public Encryption Keys

Questa SIM provides a public encryption key used by vencrypt and vhencrypt commands.

If you do not explicitly provide a public encryption key, then the commads use Questa SIM's key MGC-VERIF-SIM-RSA-2.

Mentor Graphics Public Encryption Keys

The Questa SIMsimulator provides public encryption keys to support interoperability across products. Mentor Graphics provides these encryption keys in the *keyring* directory. The files are structured as encryption envelopes providing the directives needed to use the public encryption keys.

Active Public Encryption Keys

Questa SIM provides two active public encryption keys in the installation *keyring* directory:

- *MGC-VERIF-SIM-RSA-2.active* — The supplied encryption recipe is:

```
`protect begin_toolblock
`protect key_keyowner = "Mentor Graphics Corporation"
`protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`protect key_method = "rsa"
`protect rights_digest_method="sha256"
`protect key_public_key
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtNA6tJ1tV/cXF4K5mL4s
4KCuTKWSbN/BnJJ6elRTWr2+s5Baaul0ctIX/3KYzpITmG9ph/4uZBs+jV5DAC+9
WRZQDc11JdIlRi04dEx/bGVbfPs3pdTPFZjA6gfegdW03ZNhjaJChTwEoXL1xIGP
oodJyhX9r1DoxU2lWB19vpwI5Geygh6pYgkPXb0aQzLh6hyUBhH9yMN6eV+imBbO
eax8ZCO6Gz2CJq3ebS/JoMYrikgcIEf6kVhIOiB9LluTp6TZlSd8ilwPhQmfXWH2
w4CaIpN8kADaVHnDWIdqqHlGf3cNQrlWj6FnFpSam6PjmWp5ZD4Jt6UNJxEoKEsn
gwIDAQAB
`protect end_toolblock
```

- *MGC-VERIF-SIM-RSA-3.active.* — The supplied encryption envelope is:

```
`protect begin_toolblock
`protect key_keyowner = "Mentor Graphics Corporation"
`protect key_keyname = "MGC-VERIF-SIM-RSA-3"
`protect key_method = "rsa"
`protect rights_digest_method="sha256"
`protect key_public_key
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC1hm/RxfJSXLzWIpTJWdyCDFXo
bHK1nLmxQCqPK9jjEY+cUgX90lstOWPfClj13dMOnDNkCS1+owUAiVHCXZGa/agP
gq77ioheQgXpY2kViTdgjdsjoWTIYt2ROpRO0BmJRGpXc1wT9GoFH2MYjomhNqd7jEL
fuwfMUnUAft0zXQIDAQAB
`protect end_toolblock
```

Caution

The encryption key will not work if you insert extraneous characters or spaces of any type during copy and paste operations. Please copy the encryption key from the files in the *keyring* directory, and not from this manual.

While the MGC-VERIF-SIM-RSA-2 provides the strongest protection for your code, with RSA 2048 encryption, it comes with a significant performance cost. To enable faster decoding, Questa SIM provides an encryption key of half the size, using RSA 1024 encryption. Since decryption performance varies with the cube of the key size, you can achieve a nominal 8x performance improvement with the smaller key. This public key is named MGC-VERIF-SIM-RSA-3.The use of MGC-VERIF-SIM-RSA-3 key is exactly the same as the MGC-VERIF-SIMRSA-2 key.

Deprecated Public Encryption Keys

There are two deprecated public encryption keys in the *keyring* directory. Mentor Graphics does not recommend their use. The **vencrypt** and **vhencrypt** commands will issue warning messages that they are deprecated. The following files contained deprecated keys:

- *MGC-DVT-MTI.deprecated*
- *MGC-VERIF-SIM-RSA-1.deprecated*

The presence of a file named <*keyname*>*.deprecated* in the installation subdirectory *keyring/* triggers a warning for <keyname>. These files may be managed by the CAD tools team to indicate that keys are active or deprecated.

Using Public Encryption Keys

As part of building an encryption recipe, you can use a pre-existing encryption envelope with the Mentor Graphics public key through a command line switch.

Prerequisites

- Access to theQuesta SIM *keyring* installation directory.

Procedure

1. Create the following simple D-Flip Flop with a viewport pragma exposing the value of a register. The name of the file is *flop.v*

```
`pragma protect begin
// D flip-flop
module dff (clk, reset,d, q, qb);
   input      clk;
   input      reset;
   input      d;
   output     q;
   output     qb;

`pragma protect viewport
reg        q;
assign qb = ~q;
always @(posedge clk or posedge reset)
begin
   if (reset) begin
   // Asynchronous reset when reset goes high
   q <= 1'b0;
end else begin
// Assign D to Q on positive clock edge
   q <= d;
   end
endendmodule
`pragma protect end
```

2. Invoke the following command:

   vencrypt -toolblock MGC-VERIF-SIM-RSA-2 flop.v

   The command searches the *keyring/* directory and uses the Mentor Graphics public encryption key as a default.

   Do not use the ".active" file extension on the command line. If the recipe has been deprecated, the command issues a warning. By not specifying the file type, your scripts can detect changes in the public encryption key if it were to be deprecated (the file type changed from active to deprecated).

   If you do not specify a -toolblock argument, then the **vencrypt** command uses as a default the Mentor Graphics public encryption key. You can create your own encryption recipe file using a different public encryption key. The advantage of using the -toolblock argument is that you can manage your encryption information outside your source text files. This avoids populating your source with encryption directives across many files.

3. View the resulting decryption envelope. When the command completes, it creates new file by adding a "p" to the file name suffix.

   Open file *flop.vp*.

```
`pragma protect begin_protected
`pragma protect version = 2
`pragma protect encrypt_agent = "ModelSim" , encrypt_agent_info =
"QA Baseline: 10.7 Beta - 3229913"
`pragma protect begin_commonblock
`pragma protect end_commonblock
`pragma protect begin_toolblock
`pragma protect key_keyowner = "Mentor Graphics Corporation" ,
key_keyname = "MGC-VERIF-SIM-RSA-2"
`pragma protect key_method = "rsa"
`pragma protect encoding = ( enctype = "base64" )
`pragma protect key_block
GGSPk0eai1Asw2J8kgGqP5pBizVXnLIiQp8OSVRCACaZ6YhW1AQbH1il8/PQ9f5+
uEtypFrWsDpNc4A7PUQJWmX8lYgSvzKHTRqq+UGh8PBEeJo3WftOSZCcw3sCCmGa
dCa4iH1Q0yOyq3rZsMO1Ykfg+VbNXXYFg3+SQRlBMbFzjj45E8rdvmE88uyvngF
yf9vnA/j7XEAGix4PKyESCNDn+h7bAYZTauv88E93xTl4F1wwn4eG0MNslJ6zi80
iTrtCEpLpJe1VQy8/mIliCc+75n3DWyxIs3DoJLbYFpDyIN8NSjoavpE2TSDOTln
YVmfeQqGuIFBjfNd+n+FYA==
`pragma protect rights_digest_method = "sha256"
`pragma protect end_toolblock="nH11KhiHxnz61HDLUSbihV/
gOefK4XmvaamTefH+nPY="
`pragma protect data_method = "aes128-cbc"`pragma protect encoding =
( enctype = "base64" , line_length = 64 , bytes = 560 )
`pragma protect data_block
dYqploMU2jFIbdPwp3UNVDT8KCtXsyCy0EKkf/IldGq7/3jQdOfR9PK6cspnTgvD
aUK6k2qbD18d2nPXYvKJ9O/IQLJyznoNDZJVgoH8JTArj489E/IpuswRJrn2Pdan
86K5l9kkOW7UhEJr1cwQvn88zzF5y1add03e+bUTVSiPNpeCqrQAYV/JVErzN1yd
Bw3KtRoYgrdUvV4XfXIspIlvlTL0kbiFp5P4gfHIf1WcbPbE+vjv8u3XRnONZwtv
Ngm1ag2sJawyZriUb6yMvdZLPZM2zL5s7WSzW5pf5gf7PxboANR1InH6JE1B94VN
+bAUvJ1wLFJ63nxakBPrc5ks2rJ+q6ymt3sKtYId7iQMlM5+WQnf3Sh/Y8Y8tNaS
Z40WMhBWf2Rx1TxvJ/+85ilKxIAgv1MzUOVbbDtTYnYYwCYJOuYhi/5Bc/eunomZ
ngLtAsIv/Lfi8cMat3BCe2yT0Z9SLZEZFH0q5HoMrFyqq7yyrkkpstT/YcJawbwQ
aZh5ToB2iYlJNNYV1bjLTf0rEe9oiIjxJplLilhkvByxarV9aqp7XF0388xAHKzo
dca27KFRAo3FLMrvsTbC3HlKwRiU4hXqdtF+GbJ8w5fWhWW/tsH/xiQpnX15VVbw
GpLPK4/X3aEghlhxFpfSuK9Jvap52sUVvAj91w967U15pZ1/gePp4PPncx7CVBo5
Rv3F4/kpCQEIAW7Z5CKc7YIL7LAOBMmF/MEWPpTqgn4=
`pragma protect end_protected
```

Pragmas for Specifying a Tool's Public Encryption Key

Every encryption run requires the specification of a targeted tool's public encryption key. If you do not specify a specific public key, then the encryption commands **vencrypt** and **vhencrypt** use the Questa SIM public encryption key.

The code shown below is the recipe file for the MGC-VERIF-SIM-RSA-2 key located in the *keyring* directory.

```
`protect begin_toolblock
`protect key_keyowner = "Mentor Graphics Corporation"
`protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`protect key_method = "rsa"
`protect rights_digest_method="sha256"
`protect key_public_key
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtNA6tJ1tV/cXF4K5mL4s
4KCuTKWSbN/BnJJ6elRTWr2+s5Baaul0ctIX/3KYzpITmG9ph/4uZBs+jV5DAC+9
WRZQDc11JdIlRi04dEx/bGVbfPs3pdTPFZjA6gfegdW03ZNhjaJChTwEoXL1xIGP
oodJyhX9r1DoxU2lWB19vpwI5Geygh6pYgkPXb0aQzLh6hyUBhH9yMN6eV+imBbO
eax8ZCO6Gz2CJq3ebS/JoMYrikgcIEf6kVhIOiB9LluTp6TZlSd8ilwPhQmfXWH2
w4CaIpN8kADaVHnDWIdqqHlGf3cNQrlWj6FnFpSam6PjmWp5ZD4Jt6UNJxEoKEsn
gwIDAQAB
`protect end_toolblock
```

The previous code is in VHDL syntax (`protect), but it works with the **vencrypt** tool because the tool translates the pragmas into SystemVerilog (`pragma protect) correct syntax. You can use it in conjunction with a rights file since the commands know how to process the **begin_toolblock** and **end_toolblock** directives.

The encryption recipe for a version 2 public key must be within the boundaries of a **toolbock** pragma (**begin_toolblock** and **end_toolblock**). The **toolbock** information supplied as a set of embedded pragmas at encryption initiates a transfer of information through the decryption envelope to the tool that holds the private key. The mechanism to select targeted tools uses the set of pragmas that fully describe the public key associated with a given tool. These pragmas are:

- key_keyowner
- key_keyname
- key_method
- key_public_key
- rights_digest_method

You must define only one instance of each of these pragmas for each tool in an envelope. The composite set of pragmas must be placed in a block bounded by the **begin_toolblock** and **end_toolblock** pragmas. This toolblock construct describes a key to be used at encryption that is recognized by the target tool.

The purpose of the **key_method** and **key_public_key** is to tell the encrypting tools how to encrypt the code with the key. The **key_keyowner** and **key_keyname,** as transmitted to the output text, identify the key for the decrypting tool. The rights digest method drives the signing of the assigned rights to prevent plain text tampering post-encryption.

Many of these pragmas will also be present in the post-encryption text. For example, the encryption tools transmit the key name, owner, and method pragmas unchanged. The act of encryption also consumes some pragmas and causes new ones to be inserted. For example, the

key_public_key is consumed at encryption, but a new **key_block** pragma is inserted into the post-encryption text. You should not supply a **key_block** pragma at encryption.

It is recommended that you place any tool-specific rights ("control" directives) into a separate file, and reference that file via the second part of the "-toolblock" option

Pragmas for Specifying the Session Key

For each encryption envelope you must choose a single choice of a data encryption algorithm. Questa SIM encryption tools generate a random key to be used in asymmetric encryption method called a "session key". The IP protected source code is encrypted using this session key.

The encrypting tool communicates the session key to the decrypting tool, which can be Questa SIM or some other tool, by means of a **KEY_BLOCK**.

The encrypting tool uses the **key_keyowner, key_keyname, key_method,** and **key_public_key** pragmas to encrypt and encode the session key into a **KEY_BLOCK**.

The decrypting tool reads each **KEY_BLOCK** until it finds one that specifies the key corresponding to the tool. It then decrypts the associated **KEY_BLOCK** data to determine the original session key and uses that session key to decrypt the IP source code.

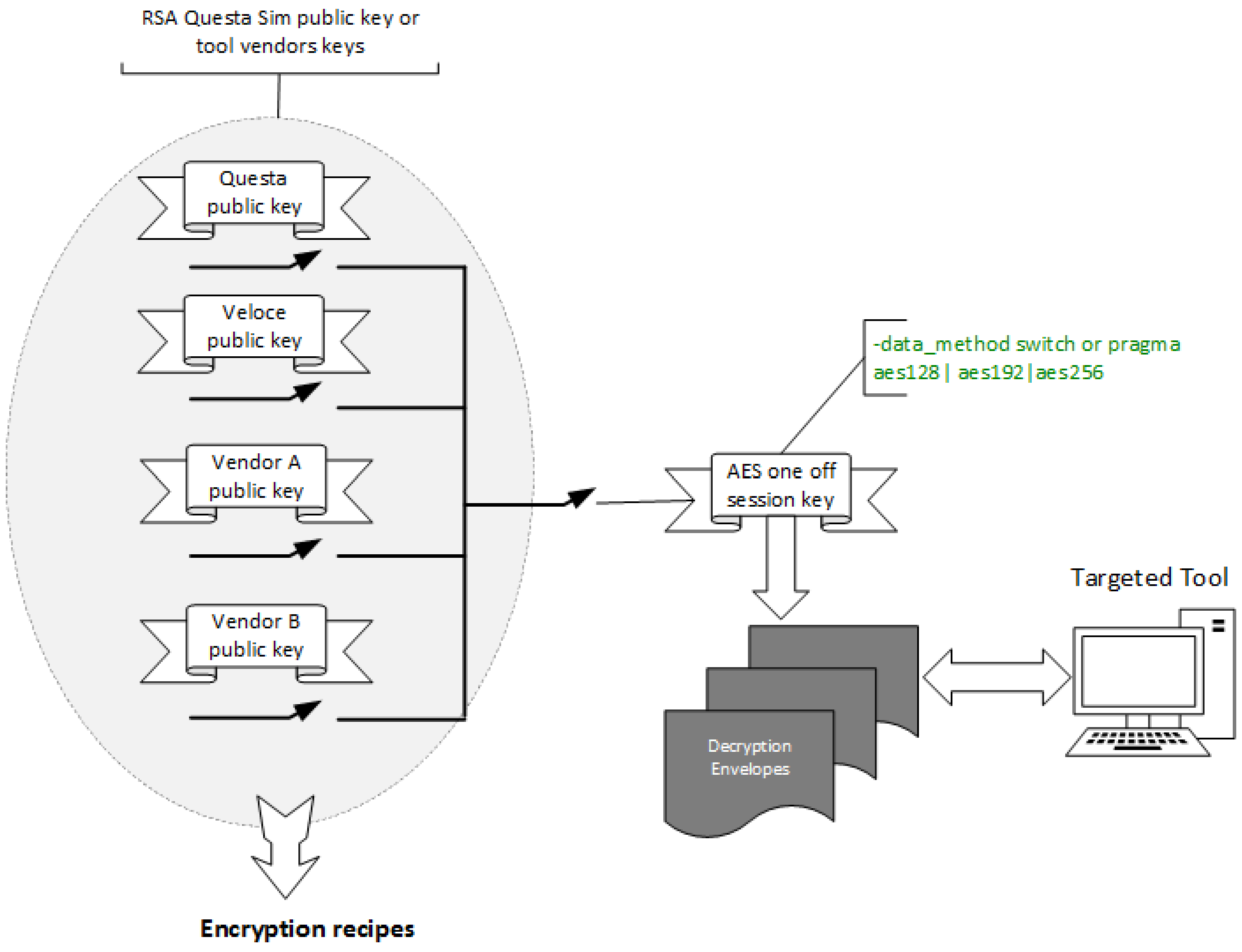
The embedded **data_method** pragma is the standard mechanism to make this choice. You can also use the -data_method command line argument. It is recommended that you verify that all targeted tools can decrypt with the algorithms specified by the **data_method** directive. All decryption tools must operate on the same data block. The encryption tools do not create a special **data_block** for each supported tool. The version 2 choices are: AES128, AES192, and AES256.

The **vencrypt** and **vhencrypt** commands provide random data required for AES key generation. As a result, you will see different **data_block** and **KEY_BLOCK** content every time you encrypt the IP.

The IP owner may optionally specify the encoding algorithm, although base64 is the recommended encoding algorithm.

Figure 1-1 shows the process of generating the session key by the encryption tools.

Figure 1-1. Session Key Usage in the Encryption Process



Deprecation Notice for the +protect vlog and vcom Argument

The command line +protect argument for vlog and vcom has been fully deprecated.

The vlog and vcom commands issue the following error message any time you use the +protect argument:

```
"**Error (suppressible): (vlog-14162) The "+protect" option to vlog is
being deprecated."
```

Mentor Graphics has moved to only one set of tools for encryption which are the **vencrypt** (and **vhencrypt** commands.

Simulation Access to Protected Objects

Protected objects are objects whose names are defined within decryption envelopes. Protection is lexical, and it is possible (but not advisable) to implement source code protection at boundaries that do not make HDL semantic sense. In these cases, simulation behavior can be undefined.

Language LRM and IEEE Expectations

The primary expectation of source code protection is that source code is not visible to the end user, but that the protected code will nonetheless compile and behave in an HDL context as if it was in its original, plaintext form.

This expected HDL behavior is not extended to tool-enabled visibilities and runtime by-name operations, including GUI, CLI, and PLI accesses. However HDL accesses using disclosed names (names that are in the protected code but of which the user has knowledge) are honored. For more detail on this, please refer to the IEEE 1735 standard.

There are some secondary expectations of protection too, one of which is that names and structures that were in the encrypted regions do not passively leak out of any of the processing tools in other ways such as tool messages. This is however balanced against cases where an IP author does something explicitly designed to emit information from under protection. As an example, a “%m” in a $display() statement in encrypted code is inherently ambiguous. The user is simultaneously requesting output of the protected module name. The tool’s behavior in many of these cases is governed by the principle that the IP author is in control from within protected code. If they choose explicitly to break the protection in this way, then they can do this.

SystemVerilog Pragmas and Smart Comment Precedence

Encryption is controlled by pragmas, and it is possible (in fact easy) to misuse pragma syntax.

In this discussion, blocking-pragma syntaxes are considered to be those where a text-block is defined by two pragmas or smart comments that establish text-start and text-end points separated by regular HDL code. For example, the following are blocking-pragma and smart-comment pairs that can enclose text blocks:

```
`ifdef and`endif
`pragma protect begin and`pragma protect end
//coverage off and //coverage on
```

SystemVerilog pragma and smart comment precedence is not well defined (and many tools have non-LRM extensions). Authors are advised to treat blocking-pragma and smart-comment syntaxes as if they mutually nest following regular nesting rules.

That is, assume that any one of these start pragmas or smart comments starts a new context. Use the paired terminal pragma/smart comment in the same context that it was started. Note that most of these constructs are not self-nesting (although this is specifically defined for each individual syntax). They can nonetheless be combined with each other in a nesting fashion:

```
`ifdef ABC
.. some text
//coverage off
.. some text
`ifdef XYZ
.. some text
`endif
```

This is a good-practice rule although not currently enforced by the language definitions. For example, it is legal but a mistake to do this in pre-encryption Verilog, where a `ifdef is in visible text but its `endif is hidden under encryption.

```
`define REALLY_BAD_IDEA_DONT_DO_THIS
`ifdef LARGE_ADDRESSES
`pragma protect begin
`define ASIZE 64
`endif
`pragma protect end
module m #ASIZE (); // etc..
```

Post-encryption this text will look something like the example below. The bad choices are now hidden under the encryption, making it very difficult for the end user to work out what is happening.

```
`define REALLY_BAD_IDEA_DONT_DO_THIS
`ifdef LARGE_ADDRESSES
`pragma protect begin_protected
(Something unreadable)
`pragma protect end_protected
module m #ASIZE (); // etc..
```

If a protected block is nested under an `ifdef that isn't currently active, you cannot guarantee that the decryption tool will decrypt the encrypted code (this is a consequence of ambiguously defined precedence). Hence in this case it will not see the `endif that terminates the original `ifdef.

Visibility Corner Cases and Ambiguities

You must be aware of potential and unexpected corner cases.

Functional Coverage

Covergroups are types and covergroup instances are dynamic objects. Dynamic objects are explicitly not covered by IEEE 1735 protection. IEEE 1735 does not mention coverage objects directly, and the covergroup model is not quite the same as other dynamic objects. Currently Questa SIM does not see a use model for protecting functional coverage since the overall purpose is to identify source items (by name and value) that have not been hit. The definition of this mode of operation is still unresolved by IEEE.

Code Coverage

Code coverage is in a similar category to functional coverage in that the purpose is to identify source-specific components to target for increased attention. Protected sources are no longer available after protection. Therefore, coverage on it is information-free as to be useless. Questa SIM does not extend code coverage into protected regions. You can create code coverage models that are not resolvable if intermediate scopes are protected.

Optimizatons

The reduced visibility into protected code has the side-effect of enabling extra optimization. The compiled model behaves as instructed by its HDL. For example, it does not have to retain hooks for logging or VPI visibility. A consequence of this can be that already-present race conditions are revealed by optimizations performed on protected code that were not performed on the unprotected original sources.

External Visibility Requests

The IEEE standard explicitly disallows tool-based runtime accesses into protected regions, even of a name that is known to be present. This prohibition includes applying command line +acc or –access to protected regions. The tools will not honor those requests.

ifdef Precedence

For IEEE 1735 version 1 and subsequent versions, the Questa SIM default precedence is that `ifdef is higher precedence than protection. That is, if there is encrypted code in an unselected `ifdef (or equivalently `else), it will not be decrypted. This can be overridden with the the vlog -svext=dbpp argument ('decrypt before preprocessing'). For version 0, the default is that the decryption operation has precedence.

Chapter 2 Questa SIM Encryption Methods

Various methods exist and options exist to help you specify the encryption results you want.

The topics covered in this section are backward compatible with previous version of the Questa SIM encryption tools.

Encryption Envelopes

Encryption envelopes define a region of textual design data or code to be protected with protection expressions. The protection expressions specify the encryption algorithm, the encryption key owner, the key name, and envelope attributes.

The beginning and ending protection expressions for Verilog/SystemVerilog are **`pragma protect begin** and **`pragma protect end**, respectively.

The beginning and ending protection expressions for VHDL are **`protect BEGIN PROTECTED** and **`protect END PROTECTED**, respectively.

The encryption envelope may contain the code to be encrypted or it may contain **`include** compiler directives that point to files containing the code to be encrypted.

You can combine symmetric and asymmetric keys in encryption envelopes to provide the safety of asymmetric keys with the efficiency of symmetric keys. The IP author can also use encryption envelopes to produce encrypted source files that can be safely decrypted by multiple authors. For these reasons, encryption envelopes are the preferred method of protection.

Creating Encryption Envelopes in Source Files

You can configure encryption envelopes to contain the actual code to encrypt or you can use `include compiler directives to point to files containing the code to encrypt.

Prerequisites

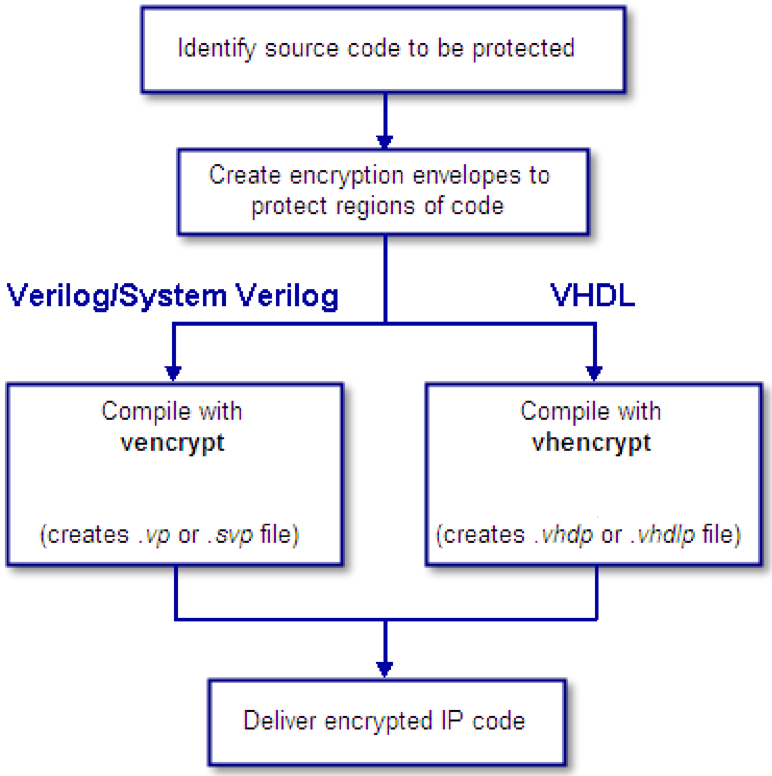
Identify the region(s) of code to be encrypted, or the files that contain the code to be encrypted.

Procedure

1. Enclose the code you want to encrypt within protection directives; or, enclose the names of the files that contain the code within protection directives.
2. Compile your code with Questa Sim the appropriate encryption utility:
   - Use the vencrypt command for Verilog and SystemVerilog design code.
   - Use the vhencrypt command for VHDL design code.

   The flow diagram for creating encryption envelopes is shown in Figure 2-1.

Figure 2-1. Create an Encryption Envelope


Identify source code to be protected
Create encryption envelopes to protect regions of code
Verilog/System Verilog
VHDL
Compile with **vencrypt**
(creates *.vp* or *.svp* file)
Compile with **vhencrypt**
(creates *.vhdp* or *.vhdlp* file)
Deliver encrypted IP code

Examples

In the example shown in Figure 2-2, Verilog design data to be encrypted follows the **`pragma protect begin** expression and ends with the **`pragma protect end** expression. If the design data had been written in VHDL, the data to be protected would follow a **`protect begin** expression and would end with a **`protect end** expression.

Figure 2-2. Encryption Envelope Contains Design Data to be Protected

```
module test_dff4(output [3:0] q, output err);
   parameter WIDTH = 4;
   parameter DEBUG = 0;
   reg [3:0] d;
   reg   clk;

   dff4 d4(q, clk, d);

   assign   err = 0;

   initial
     begin
      $dump_all_vpi;
      $dump_tree_vpi(test_dff4);
      $dump_tree_vpi(test_dff4.d4);
      $dump_tree_vpi("test_dff4");
      $dump_tree_vpi("test_dff4.d4");
      $dump_tree_vpi("test_dff4.d", "test_dff4.clk", "test_dff4.q");
      $dump_tree_vpi("test_dff4.d4.d0", "test_dff4.d4.d3");
      $dump_tree_vpi("test_dff4.d4.q", "test_dff4.d4.clk");
     end
endmodule

module dff4(output [3:0] q, input clk, input [3:0] d);
`pragma protect data_method = "aes128-cbc"
`pragma protect author = "IP Provider"
`pragma protect author_info = "Widget 5 version 3.2"
`pragma protect key_keyowner = "Mentor Graphics Corporation"
`pragma protect key_method = "rsa"
`pragma protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`pragma protect begin
   dff_gate d0(q[0], clk, d[0]);
   dff_gate d1(q[1], clk, d[1]);
   dff_gate d2(q[2], clk, d[2]);
   dff_gate d3(q[3], clk, d[3]);
endmodule // dff4

module dff_gate(output q, input clk, input d);
   wire preset = 1;
   wire clear = 1;

   nand #5
     g1(l1,preset,l4,l2),
     g2(l2,l1,clear,clk),
     g3(l3,l2,clk,l4),
     g4(l4,l3,clear,d),
     g5(q,preset,l2,qbar),
     g6(qbar,q,clear,l3);
endmodule
`pragma protect end
```

In the code shown in Figure 2-3, design data is contained in three files - *diff.v*, *prim.v*, and *top.v*. This shows how to configure the encryption envelope so the entire contents of *diff.v*, *prim.v*, and *top.v* are encrypted.

Figure 2-3. Encryption Envelope Contains `include Compiler Directives

```
`timescale 1ns / 1ps
`cell define

module dff (q, d, clear, preset, clock);
output q;
input d, clear, preset, clock;
reg q;

`pragma protect data_method = "aes128-cbc"
`pragma protect author = "IP Provider", author_info = "Widget 5 v3.2"
`pragma protect key_keyowner = "Mentor Graphics Corporation"
`pragma protect key_method = "rsa"
`pragma protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`pragma protect begin

`include diff.v
`include prim.v
`include top.v

`pragma protect end

always @(posedge clock)
   q = d;

endmodule

`endcelldefine
```

The `include Compiler Directive (Verilog only)

You can use the ‘include directive in Verilog files during encryption.

If any **`include** directives occur within a protected region of Verilog code, the encryption commands generates a copy of the include file with a “*.vp*” or a “*.svp*” extension and encrypts the entire contents of the include file.

Consider the following header file, *header.v*, consisting of the following source code:

```
initial begin
   a <= b;
   b <= c;
end
```

and the file you want to encrypt, *top.v*, contains the following source code:

```
module top;
   `pragma protect begin
   `include "header.v"
   `pragma protect end
endmodule
```

then, when you use the encryption commands, they encrypt the source code of the header file. If you could decrypt the resulting *work/top.vp* file it would look like:

```
module top;
   `pragma protect begin
   initial begin
      a <= b;
      b <= c;
   end
   `pragma protect end
endmodule
```

When you use the vencrypt compile utility (see Delivering IP Code with Undefined Macros), it will treat any **`include** statements as text just like any other source code and it will encrypt them with the other Verilog/SystemVerilog source code. So, if you use the vencrypt utility on the *top.v* file above, the resulting *work/top.vp* file would look like the following (if we could decrypt it):

```
module top;
   `protect
   `include "header.v"
   `endprotect
endmodule
```

The vencrypt utility will not create an encrypted version of *header.h*.

You can avoid such errors by creating a dummy module that includes the parameter declarations. For example, if you have a file that contains your parameter declarations and a file that uses those parameters, you can do the following:

```
module dummy;
   `protect
   `include "params.v" // contains various parameters
   `include "tasks.v" // uses parameters defined in params.v
   `endprotect
endmodule
```

After encryption, the work library contains encrypted versions of *params.v* and *tasks.v*, called *params.vp* and *tasks.vp*. You can then copy these encrypted files out of the work directory to more convenient locations.You can then include these encrypted files within your design. For example:

```
module main
'include "params.vp"
'include "tasks.vp"
        ...
```

Portable Encryption for Multiple Tools

An IP author can use the concept of multiple key blocks to produce code that is secure and portable across any tool that supports Version 1 recommendations from the IEEE P1735 working group. This capability is not language-specific—you can use it for VHDL or Verilog.

For example, suppose the author wants to modify the following VHDL *sample file* so the encrypted model can be decrypted and simulated by both Questa Sim and by a hypothetical company named XYZ inc.

```
========== sample file ==========

-- The entity "ip1" is not protected
...
entity ip1 is
...
end ip1;

-- The architecture "a" is protected
-- The internals of "a" are hidden from the user
`protect data_method = "aes128-cbc"
`protect encoding = ( enctype = "base64" )
`protect key_keyowner = "Mentor Graphics Corporation"
`protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`protect key_method = "rsa"
`protect KEY_BLOCK
`protect begin
architecture a of ip1 is
...
end a;
`protect end

-- Both the entity "ip2" and its architecture "a" are completely protected
`protect data_method = "aes128-cbc"
`protect encoding = ( enctype = "base64" )
`protect key_keyowner = "Mentor Graphics Corporation"
`protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`protect key_method = "rsa"
`protect KEY_BLOCK
`protect begin
library ieee;
use ieee.std_logic_1164.all;
entity ip2 is
...
end ip2;
architecture a of ip2 is
...
end a;
`protect end

========== end of sample file ==========
```

The author does this by writing a key block for each decrypting tool. If XYZ publishes a public key, the two key blocks in the IP source code might look like the following:

```
`protect key_keyowner = "Mentor Graphics Corporation"
`protect key_method = "rsa"
`protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`protect KEY_BLOCK
`protect key_keyowner = "XYZ inc"
`protect key_method = "rsa"
`protect key_keyname = "XYZ-keyPublicKey"
`protect key_public_key = <public key of XYZ inc.>
`protect KEY_BLOCK
```

The encrypted code would look very much like the *sample file*, with the addition of another key block:

```
`protect key_keyowner = "XYZ inc"
`protect key_method = "rsa"
`protect key_keyname = "XYZ-keyPublicKey"
`protect KEY_BLOCK
   <encoded encrypted key information for "XYZ inc">
```

Questa Sim uses its key block to determine the encrypted session key and XYZ Incorporated uses the second key block to determine the same key. Consequently, both implementations could successfully decrypt the code.

Note

The IP owner is responsible for obtaining the appropriate key for the specific tool(s) protected IP is intended for. The author should also validate the encrypted results with those tools to ensure the IP is protected and will function as intended in those tools.

Language-Specific Usage Models

Usage models for protecting your source code are language-specific.

Usage Models for Protecting Verilog Source Code

The encryption capabilities of Questa Sim support Verilog and SystemVerilog usage models for IP authors and their customers.

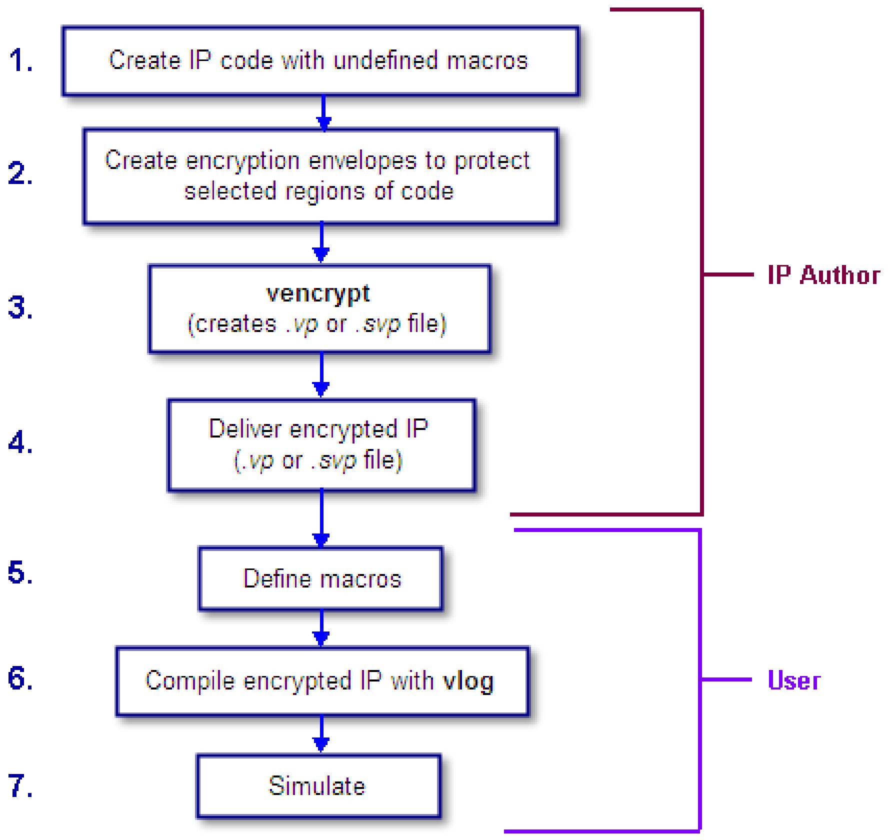
- IP authors may use the vencrypt utility to deliver Verilog and SystemVerilog code containing *undefined* macros and `directives. The IP user can then define the macros and `directives and use the code in a wide range of EDA tools and design flows. See Delivering IP Code with Undefined Macros.
- IP authors can use **`pragma protect** directives to protect Verilog and SystemVerilog code containing *user-defined* macros and `directives. Authors can deliver the IP code to IP customers for use in a wide range of EDA tools and design flows. See Delivering IP Code with User-Defined Macros.

Delivering IP Code with Undefined Macros

The vencrypt utility enables IP authors to deliver VHDL and Verilog/ SystemVerilog IP code (respectively) that contains undefined macros and `directives. End users can then deploy the resulting encrypted IP code in a wide range of EDA tools and design flows.

The recommended encryption usage flow is shown in Figure 2-4.

Figure 2-4. Verilog/SystemVerilog Encryption Usage Flow


1.
Create IP code with undefined macros
2.
Create encryption envelopes to protect selected regions of code
3.
vencrypt (creates .vp or .svp file)
4.
Deliver encrypted IP (.vp or .svp file)
5.
Define macros
6.
Compile encrypted IP with vlog
7.
Simulate
IP Author
User

Procedure

1. The IP author creates code that contains undefined macros and `directives.
2. The IP author creates encryption envelopes to protect selected regions of code or entire files.
3. The IP author uses the vencrypt utility to encrypt Verilog and SystemVerilog code contained within encryption envelopes. Macros are not pre-processed before encryption, so macros and other `directives are unchanged.

   The vencrypt utility produces a file with a *.vp* or a *.svp* extension to distinguish it from non-encrypted Verilog and SystemVerilog files, respectively. You can change the file extension for use with simulators other than Questa Sim. The original file extension is preserved if you specify the -d <dirname> argument with vencrypt, or if you use a `directive in the file to be encrypted.

   The IP author can use the -h <filename> argument for vencrypt to specify a header file to encrypt a large number of files that do not contain the **`pragma protect** or proprietary **`protect** information—about how to encrypt the file. Instead, encryption information is provided in the <filename> specified by -h <filename>. This argument concatenates the header file onto the beginning of each file so that you do not have to manually edit each file to add the same **`pragma protect**.

For example:

vencrypt -h encrypt_head top.v cache.v gates.v memory.v

concatenates the information in the *encrypt_head* file into each Verilog file listed. The *encrypt_head* file may look like the following:

```
`pragma protect data_method = "aes128-cbc"
`pragma protect author = "IP Provider"
`pragma protect key_keyowner = "Mentor Graphics Corporation"
`pragma protect key_method = "rsa"
`pragma protect key_keyname = "MGC-VERIF-SIM-RSA-1"
`pragma protect encoding = (enctype = "base64")
`pragma protect begin
```

Notice that there is no **`pragma protect end** expression in the header file, just the header block that starts the encryption. The **`pragma protect end** expression is implied by the end of the file.

4. The IP author delivers encrypted IP with undefined macros and `directives.
5. The IP user defines macros and `directives.
6. The IP user compiles the design with vlog.
7. The IP user simulates the design with Questa Sim or other simulation tools.

Delivering IP Code with User-Defined Macros

IP authors can use **`pragma protect** expressions to protect proprietary code containing user-defined macros and `directives. End users can then deploy the resulting encrypted IP code in a wide range of EDA tools and design flows.

Procedure

1. The IP author creates proprietary code that contains user-defined macros and `directives.
2. The IP author creates encryption envelopes with **`pragma protect** expressions to protect regions of code or entire files.
3. The IP author uses the encryption commands to encrypt IP code contained within encryption envelopes.
4. (Optional) You can change the *.vp* or *.vp* extension of the encryted file so that the file can be used by a simulator other than Questa Sim. The original file extension is preserved if a `directive is used in the file to be encrypted.
5. The IP author delivers the encrypted IP.
6. The IP user simulates the code like any other file.

Results

When encrypting source text, any macros without parameters defined on the command line are substituted (not expanded) into the encrypted file. This makes certain macros unavailable in the encrypted source text.

Questa Sim takes every simple macro that is defined with the compile command (vlog) and substitutes it into the encrypted text. This prevents third party users of the encrypted blocks from having access to or modifying these macros.

Note

Macros not specified with vlog via the **+define+** option are unmodified in the encrypted block.

Examples

For example, the code below is an example of a file that might be delivered by an IP provider. The filename for this module is *example00.sv*

```
`pragma protect data_method = "aes128-cbc"
`pragma protect key_keyowner = "Mentor Graphics Corporation"
`pragma protect key_method = "rsa"
`pragma protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`pragma protect author = "Mentor", author_info = "Mentor_author"
`pragma protect begin
`timescale  1 ps / 1 ps

module example00 ();
    `ifdef IPPROTECT
        reg `IPPROTECT ;
        reg otherReg ;
        initial begin
        `IPPROTECT  = 1;
        otherReg    = 0;

        $display("ifdef defined as true");

        `define FOO 0
        $display("FOO is defined as: ", `FOO);
        $display("reg IPPROTECT has the value: ", `IPPROTECT );
        end
    `else
        initial begin
        $display("ifdef defined as false");
        end
    `endif

endmodule

`pragma protect end
```

After invoking vencrypt his creates an encrypted file called *encrypted00.sv*. You can then compile this file with a macro override for the macro “FOO” as follows:

```
vlog +define+FOO=99 encrypted00.sv
```

A customer then can override the macro FOO while the macro IPPROTECT retains the value specified at the time of encryption, and the macro IPPROTECT no longer exists in the encrypted file.

Usage Models for Protecting VHDL Source Code

Encryption capabilities for VHDL are supported for a number of usage models.

Supported usage models include:

- IP authors can use `**protect** directives to create an encryption envelope (see Encryption Envelopes) for the VHDL code to be protected and use Questa Sim's vhencrypt utility to encrypt the code. The encrypted IP code can be delivered to IP customers for use in a wide range of EDA tools and design flows. See Using the vhencrypt Utility.
- IP authors can use `**protect** directives to create an encryption envelope (see Encryption Envelopes) for the VHDL code to be protected and use Questa Sim's default encryption and decryption actions. The IP code can be delivered to IP customers for use in a wide range of EDA tools and design flows. See Questa Sim Default Encryption for VHDL.
- IP authors can use `**protect** directives to create an encryption envelope for VHDL code and select encryption methods and encoding other than Questa Sim's default methods. See User-Selected Encryption for VHDL.
- IP authors can use "raw" encryption and encoding to aid debugging. See Raw Encryption for VHDL.
- IP authors can encrypt several parts of the source file, choose the encryption method for encrypting the source (the data_method), and use a key automatically provided by Questa Sim. See Encryption of Several Parts of a VHDL Source File.
- IP authors can use the concept of multiple key blocks to produce code that is secure and portable across different simulators. See Portable Encryption for Multiple Tools.

Note

VHDL encryption requires that the KEY_BLOCK (the sequence of key_keyowner, key_keyname, and key_method directives) end with a `**protect KEY_BLOCK** directive.

Using the vhencrypt Utility

The vhencrypt utility enables IP authors to deliver encrypted VHDL IP code to users. The resulting encrypted IP code can then be used in a wide range of EDA tools and design flows.

Procedure

1. The IP author creates code.
2. The IP author creates encryption envelopes (see Encryption Envelopes) to protect selected regions of code or entire files.

3. The IP author uses the Questa Simvhencrypt utility to encrypt code contained within encryption envelopes.

   The vhencrypt utility produces a file with a *.vhdp* or a *.vhdlp* extension to distinguish it from non-encrypted VHDL files.

4. (Optional) Change the file extension for use with simulators other than Questa Sim.

   The original file extension is preserved if the -d <dirname> argument is used with vhencrypt.

5. (Optional) use the -h <filename> argument for vencrypt the IP author may specify a header file that can be used to encrypt a large number of files that do not contain the **`protect** information about how to encrypt the file.

   Instead, encryption information is provided in the <filename> specified by -h <filename>. This argument essentially concatenates the header file onto the beginning of each file and saves the user from having to edit hundreds of files in order to add in the same **`protect** to every file.

   For example:

   vhencrypt -h encrypt_head top.vhd cache.vhd gates.vhd memory.vhd

   concatenates the information in the *encrypt_head* file into each VHDL file listed. The *encrypt_head* file may look like the following:

   ```
   `protect data_method = "aes128-cbc"
   `protect author = "IP Provider"
   `protect encoding = (enctype = "base64")
   `protect key_keyowner = "Mentor Graphics Corporation"
   `protect key_method = "rsa"
   `protect key_keyname = "MGC-VERIF-SIM-RSA-2"
   `protect KEY_BLOCK
   `protect begin
   ```

   Notice that there is no **`protect end** expression in the header file, just the header block that starts the encryption. The **`protect end** expression is implied by the end of the file.

6. The IP author delivers encrypted IP.

7. The IP user compiles the design with vcom.

8. The IP user simulates the design with Questa Sim or other simulation tools.

Examples

Questa Sim Default Encryption for VHDL

Suppose an IP author needs to make a design entity, called IP1, visible to the user so the user can instantiate the design, but the author wants to hide the architecture implementation from the user. In addition, suppose that IP1 instantiates entity IP2, which the author wants to hide completely from the user. The easiest way to accomplish this is to surround the regions to be protected with **`protect begin** and **`protect end** directives and let Questa Sim choose default actions. For this example, all the source code exists in a single file, *example1.vhd*:

```
========== file example1.vhd ==========
-- The entity "ip1" is not protected
...
entity ip1 is
...
end ip1;
-- The architecture "a" is protected
-- The internals of "a" are hidden from the user
`protect begin
architecture a of ip1 is
...
end a;
`protect end
-- Both the entity "ip2" and its architecture "a" are completely protected
`protect begin
entity ip2 is
...
end ip2;
architecture a of ip2 is
...
end a;
`protect end
========== end of file example1.vhd ==========
```

The compiler produces an encrypted file, *example1.vhdp* which looks like the following:

```
========== file example1.vhdp ==========
-- The entity "ip1" is not protected
...
entity ip1 is
...
end ip1;
-- The architecture "a" is protected
-- The internals of "a" are hidden from the user
`protect BEGIN_PROTECTED
`protect version = 1
`protect encrypt_agent = "Model Technology", encrypt_agent_info = "DEV"
`protect key_keyowner = "Mentor Graphics Corporation"
`protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`protect key_method = "rsa"
`protect encoding = ( enctype = "base64" )
`protect KEY_BLOCK
  <encoded encrypted session key>
`protect data_method="aes128-cbc"
`protect encoding = ( enctype = "base64" , bytes = 224 )
`protect DATA_BLOCK
   <encoded encrypted IP>
`protect END_PROTECTED
-- Both the entity "ip2" and its architecture "a" are completely protected
`protect BEGIN_PROTECTED
`protect version = 1
`protect encrypt_agent = "Model Technology", encrypt_agent_info = "DEV"
`protect key_keyowner = "Mentor Graphics Corporation"
`protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`protect key_method = "rsa"
`protect encoding = ( enctype = "base64" )
`protect KEY_BLOCK
   <encoded encrypted session key>
`protect data_method = "aes128-cbc"
`protect encoding = ( enctype = "base64" , bytes = 224 )
`protect DATA_BLOCK
   <encoded encrypted IP>
`protect END_PROTECTED
========== end of file example1.vhdp ==========
```

When the IP author surrounds a text region using only **`protect begin** and **`protect end**, Questa Sim uses default values for both encryption and encoding. The first few lines following the **`protect BEGIN_PROTECTED** region in file *example1.vhdp* contain the key_keyowner, key_keyname, key_method and KEY_BLOCK directives. The session key is generated into the key block, and that key block is encrypted using the “rsa” method. The data_method indicates that the default data encryption method is aes128-cbc, and the “enctype” value shows that the default encoding is base64.

Alternatively, the IP author can compile file *example1.vhd* with the command:

vhencrypt example1.vhd

Here, the author does not supply the name of the file to contain the protected source. Instead, vhencryp creates a protected file, gives it the name of the original source file with a 'p' placed at the end of the file extension, and puts the new file in the current work library directory. With the command described above, vhencrypt creates file *work/example1.vhdp*.

In Questa Sim, default encryption methods provide an easy way for IP authors to encrypt VHDL designs while hiding the architecture implementation from the user. The results are usable only by Questa Sim tools.

User-Selected Encryption for VHDL

Suppose that the IP author wants to reproduce the code shown in the *example1.vhd* file used above, but wants to provide specific values and not use any default values. To do this the author adds **`protect** directives for keys, encryption methods, and encoding, and places them before each **`protect begin** directive. The input file would look like the following:

```
========== file example2.vhd ==========
-- The entity "ip1" is not protected
...
entity ip1 is
...
end ip1;
-- The architecture "a" is protected
-- The internals of "a" are hidden from the user
`protect data_method = "aes128-cbc"
`protect encoding = ( enctype = "base64" )
`protect key_keyowner = "Mentor Graphics Corporation"
`protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`protect key_method = "rsa"
`protect KEY_BLOCK
`protect begin
architecture a of ip1 is
...
end a;
`protect end
-- Both the entity "ip2" and its architecture "a" are completely protected
`protect data_method = "aes128-cbc"
`protect encoding = ( enctype = "base64" )
`protect key_keyowner = "Mentor Graphics Corporation"
`protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`protect key_method = "rsa"
`protect KEY_BLOCK
`protect begin
library ieee;
use ieee.std_logic_1164.all;
entity ip2 is
...
end ip2;
architecture a of ip2 is
...
end a;
`protect end
========== end of file example2.vhd ==========
```

The data_method directive indicates that the encryption algorithm “aes128-cbc” should be used to encrypt the source code (data). The encoding directive selects the “base64” encoding method, and the various key directives specify that the Mentor Graphic key named “MGC-VERIF-SIM-

RSA-2" and the "RSA" encryption method are to be used to produce a key block containing a randomly generated session key to be used with the "aes128-cbc" method to encrypt the source code. See Mentor Graphics Public Encryption Keys.

Raw Encryption for VHDL

Suppose that the IP author wants to use "raw" encryption and encoding to help with debugging the following entity:

```
entity example3_ent is    port (
    in1  : in  bit;
    out1 : out bit);
end example3_ent;
```

Then the architecture the author wants to encrypt might be this:

```
========== File example3_arch.vhd
`protect data_method = "raw"
`protect encoding = ( enctype = "raw")
`protect begin
architecture arch of example3_ent is begin out1 <= in1 after 1 ns;  end
arch;
`protect end
========== End of file example3_arch.vhd ==========
```

If (after compiling the entity) the *example3_arch.vhd* file were encrypted using the command:

vhencrypt example3_arch.vhd

Then the following file would be produced in the work directory

```
========== File work/example3_arch.vhdp ==========
`protect data_method = "raw"
`protect encoding = ( enctype = "raw")
`protect BEGIN_PROTECTED
`protect version = 1
`protect encrypt_agent = "Model Technology", encrypt_agent_info = "DEV"
`protect data_method = "raw"
`protect encoding = ( enctype = "raw", bytes = 81 )
`protect DATA_BLOCK
architecture arch of example3_ent is
begin
out1 <= in1 after 1 ns;
end arch;
`protect END_PROTECTED
========== End of file work/example3_arch.vhdp
```

Notice that the protected file is very similar to the original file. The differences are that **`protect begin** is replaced by **`protect BEGIN_PROTECTED**, **`protect end** is replaced by **`protect END_PROTECTED**, and some additional encryption information is supplied after the **BEGIN PROTECTED** directive.

Encryption of Several Parts of a VHDL Source File

This example shows the use of symmetric encryption. It also demonstrates another common use model, in which the IP author encrypts several parts of a source file, chooses the encryption method for encrypting the source code (the **data_method**), and uses a key automatically provided by Questa Sim. (This is very similar to the proprietary **`protect** method in Verilog - see Proprietary Source Code Encryption Tools.)

```
========== file example4.vhd ==========
entity ex4_ent is
end ex4_ent;
architecture ex4_arch of ex4_ent is
  signal s1: bit;
`protect data_method = "aes128-cbc"
`protect begin
  signal s2: bit;
`protect end
  signal s3: bit;
begin  -- ex4_arch
`protect data_method = "aes128-cbc"
`protect begin
s2 <= s1 after 1 ns;
`protect end
s3 <= s2 after 1 ns;
end ex4_arch;
========== end of file example4.vhd
```

If this file were encrypted using the command:

```
vhencrypt example4.vhd
```

Then the following file would be produced in the work directory:

```
========== File work/example4.vhdp ==========
entity ex4_ent is
end ex4_ent;
architecture ex4_arch of ex4_ent is
  signal s1: bit;
`protect data_method = "aes128-cbc"
`protect BEGIN_PROTECTED
`protect version = 1
`protect encrypt_agent = "Model Technology", encrypt_agent_info = "DEV"
`protect data_method = "aes128-cbc"
`protect encoding = ( enctype = "base64" , bytes = 18 )
`protect DATA_BLOCK
<encoded encrypted declaration of s2>
`protect END_PROTECTED
  signal s3: bit;
begin  -- ex4_arch
`protect data_method = "aes128-cbc"
`protect BEGIN_PROTECTED
`protect version = 1
`protect encrypt_agent = "Model Technology", encrypt_agent_info = "DEV"
`protect data_method = "aes128-cbc"
`protect encoding = ( enctype = "base64" , bytes = 21 )
`protect DATA_BLOCK
<encoded encrypted signal assignment to s2>
`protect END_PROTECTED
s3 <= s2 after 1 ns;
end ex4_arch;
========== End of file work/example4.vhdp
```

The encrypted *example4.vhdp* file shows that an IP author can encrypt both declarations and statements. Also, note that the signal assignment

```
s3 <= s2 after 1 ns;
```

is not protected. This assignment compiles and simulates even though signal s2 is protected. In general, executable VHDL statements and declarations simulate the same whether or not they refer to protected objects.

Proprietary Source Code Encryption Tools

Mentor Graphics provides two proprietary methods for encrypting source code.

- The **`protect** / **`endprotect** compiler directives allow you to encrypt regions within Verilog and SystemVerilog files.
- The -nodebug argument for the vcom and vlog compile commands allows you to encrypt entire VHDL, Verilog, or SystemVerilog source files.

Using Proprietary Compiler Directives

The proprietary **`protect** vlog compiler directive is not compatible with other simulators. Though other simulators have a **`protect** directive, the algorithm Questa Sim uses to encrypt Verilog and SystemVerilog source files is different. Therefore, even though an uncompiled source file with **`protect** is compatible with another simulator, once the source is compiled in Questa Sim, the resulting *.vp* or *.svp* source file is not compatible.

Note
While Questa Sim supports both **`protect** and **`pragma protect** encryption directives, these two approaches to encryption are incompatible. Code encrypted by one type of directive cannot be decrypted by another.

Procedure

1. The IP author protects selected regions of Verilog or SystemVerilog IP with the **`protect** / **`endprotect** directive pair. The code in **`protect** / **`endprotect** encryption envelopes has all debug information stripped out. This behaves exactly as if using

   ```
   vlog -nodebug=ports+pli
   ```

   except that it applies to selected regions of code rather than the whole file.

2. The IP author uses the vencrypt and vhencrypt commands to encrypt IP code. Use the vencrypt utility if the code also contains undefined macros or `directives, but the code must then be compiled and simulated with Questa Sim.

3. Copy the original source file to a new file in the current work directory.

   The encryption commands produces a *.vp* or a *.svp* extension to distinguish it from other non-encrypted Verilog and SystemVerilog files, respectively. For example, *top.v* becomes *top.vp* and *cache.sv* becomes *cache.svp*.

4. Deliver the new file to be used as a replacement for the original source file.

Protecting Source Code Using -nodebug

You or IP authors can use the proprietary vlog -nodebug or vcom -nodebug command to protect entire files. The -nodebug argument for both vcom and vlog hides internal model data, allowing you to provide pre-compiled libraries without providing source code and without revealing internal model variables and structure.

Prerequisites

Identify files to be encrypted.

Note

The -nodebug argument encrypts entire files. The **`protect** compiler directive allows you to encrypt regions within a file.

Procedure

1. Compile VHDL files to be encrypted with the vcom -nodebug command.
2. Compile Verilog/SystemVerilog files to be encrypted with the vlog -nodebug command.

   When you compile with -nodebug, all source text, identifiers, and line number information are stripped from the resulting compiled object, so Questa Sim cannot locate or display any information of the model except for the external pins.

   You can access the design units that constitute your model by using the library, and you can invoke vsim directly on any of these design units to see the ports. To restrict even this access in the lower levels of your design, you can use the following -nodebug options when you compile:

Table 2-1. Compile Options for the -nodebug Compiling

Command and Switch	Result
vcom -nodebug=ports	makes the ports of a VHDL design unit invisible
vlog -nodebug=ports	makes the ports of a Verilog design unit invisible
vlog -nodebug=pli	prevents the use of PLI functions to interrogate the module for information
vlog -nodebug=ports+pli	combines the functions of -nodebug=ports and -nodebug=pli

Tip

Do not use the =ports option on a design without hierarchy, or on the top level of a hierarchical design. If you do, no ports will be visible for simulation. Rather, compile all lower portions of the design with -nodebug=ports first, then compile the top level with -nodebug alone.

Note

Design units or modules compiled with -nodebug can only instantiate design units or modules that are also compiled -nodebug.

Do not use -nodebug=ports for mixed language designs, especially for Verilog modules to be instantiated inside VHDL.

Do not use -nodebug=ports when the parent is part of a vopt -pdu (black-box) flow or for mixed language designs, especially for Verilog modules to be instantiated inside VHDL.

The Runtime Encryption Model

Encypting with the vencypt and vhencypt commands hide all source text, identifiers, and line number information from the end user in the resulting file.

Due to this behavior, Questa Sim cannot locate or display any information of the encrypted regions. Specifically, this means the following:

- The Source window will not display the source code of the design units.
- The Structure window will not display the internal structure.
- The Objects window will not display internal signals.
- The Processes window will not display internal processes.
- The Locals window will not display internal variables.
- You cannot access any of the hidden objects through the Dataflow or Schematic window or with Questa Sim commands.

Getting Information on Decryption Envelopes in Active Designs

Generate a report on decryption envelopes in a simulated design.

Prerequisites

- Your simulation uses encrypted files with decryption envelopes.

Procedure

1. Invoke the following command to view information on every decryption envelope loaded in QuestaSIM:

   ```
   report protected envelopes
   ```

2. Invoke the report protected envelopes command again, this time with an id argument,

   ```
   report protected envelopes 1
   ```

Results

The command produces the following output assuming two decryption envelopes in the entire design.

```
report protected envelopes
```

```
#
# SessionId = 1
#       HMACId = 17omXt5ndpIl9ehHiCX2UgTmvjyIDcb4OYHkAhA3bqU=
#       File = prot.svp
#       Line = 9
#       Version = 2
#       Right decryption = "true"
#       Right runtime_visibility = "delegated"
#       Right child_visibility = "delegated"
#       Right error_handling(note) = "plaintext"
#       Right error_handling(warning) = "srcrefs"
#       Right error_handling(error) = "srcrefs"
#       Right error_handling(fatal) = "srcrefs"
#       License Proxy Name = myproxy
#       License Key Owner = Mentor Graphics Corporation
#       License Key Name = MGC-VERIF-SIM-RSA-1
#       License Symmetric Key Method = aes128-cbc
#       License Public Key Method = rsa
#       License Public Key =
# MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCnJfQb+LLzTMX3NRARsv7A8+LV
# 5SgMEJCvIf9Tif2emi4z0qtp8E+nX7QFzocTlClC6Dcq2qIvEJcpqUgTTD+mJ6gr
# JSJ+R4AxxCgvHYUwoT80Xs0QgRqkrGYxW1RUnNBcJm4ZULexYz8972Oj6rQ99n5e
# 1kDa/eBcszMJyOkcGQIDAQAB
#       License Atributes = USER,MAC,PROXYINFO=1.2
#
# # SessionId = 2
#       HMACId = nWBKalWRo8Cb+CNwTRYgNHHj5ZYjxlVvr8vVN9MoAZw=
#       File = prot.svp
#       Line = 97
#       Version = 2
#       Right decryption = "delegated"
#       Right runtime_visibility = "delegated"
#       Right child_visibility = "delegated"
#       Right error_handling(note) = "delegated"
#       Right error_handling(warning) = "delegated"
#       Right error_handling(error) = "srcrefs"
#       Right error_handling(fatal) = "delegated"
#       License Proxy Name = myproxy
#       License Key Owner = Mentor Graphics Corporation
#       License Key Name = MGC-VERIF-SIM-RSA-2
#       License Symmetric Key Method = aes128-cbc
#       License Public Key Method = rsa
#       License Public Key =
# MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCnJfQb+LLzTMX3NRARsv7A8+LV
# 5SgMEJCvIf9Tif2emi4z0qtp8E+nX7QFzocTlClC6Dcq2qIvEJcpqUgTTD+mJ6gr
# JSJ+R4AxxCgvHYUwoT80Xs0QgRqkrGYxW1RUnNBcJm4ZULexYz8972Oj6rQ99n5e
# 1kDa/eBcszMJyOkcGQIDAQAB
#       License Attributes = <NULL>
```

You can record the individual IDs for later use. The report has a file/line location as well as some properties (values) based on the original protected region.The file and line listing is where the decryption envelope begins in the referenced file.

Each rights entry shows the value of that right. If there is a license proxy as part of the decryption envelope, then the license labels will show the values of each of the strings. These strings are also present in the common block of the decryption envelope.

Chapter 3
IEEE 1735 Supported Encryption Methods

The IEEE 1735 is the standard for encryption interoperability.

It addresses use models, algorithm choices, conventions, and minor corrections to the HDL standards to achieve useful interoperability.The IEEE 1735 standard implies various encryption methods that differ from previous encryption standards for IP.

Source File Encryption Methods

There are two basic encryption methods. The first method involves embedding encryption recipes in the source files. The second method is to externalize all encryption recipes into their own files. This is performed by separating common and tool blocks away from the original the source file.

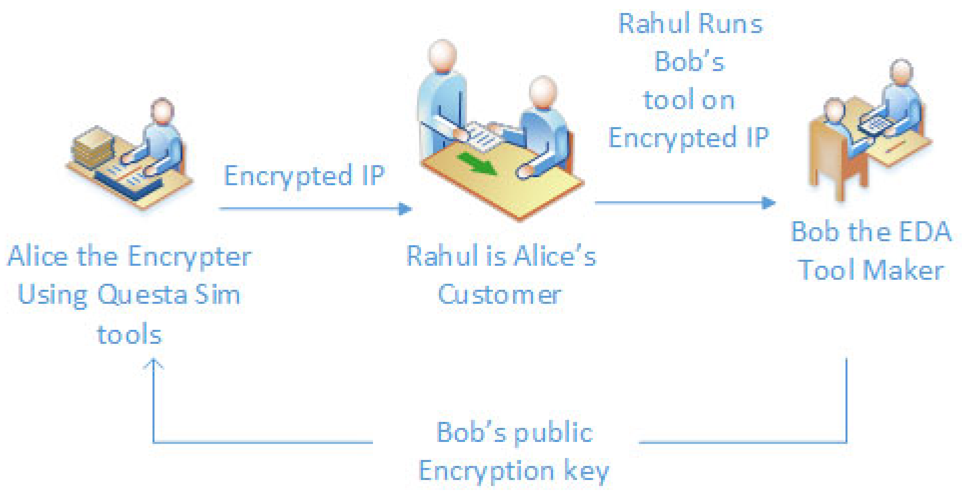
To understand the encryption methods, you should understand the main actors in an encryption process.

Basic Model of Encryption Users

To discuss encryption methods, the section uses the fictional characters Alice, Rahul, and Bob. The names Alice and Bob have been commonly used as place holders in cryptological literature since 1978. A third character has been added named "Rahul".

Figure 3-1shows the basic relationship between Alice, Rahul, and Bob in the Questa SIM encryption flow.

Figure 3-1. Alice, Rahul, and Bob


Rahul Runs Bob's tool on Encrypted IP
Encrypted IP
Alice the Encrypter Using Questa Sim tools
Rahul is Alice's Customer
Bob the EDA Tool Maker
Bob's public Encryption key

Alice, Rahul and Bob, tasks, and responsibilities include the following:

- *Alice* is a senior member of an IP development team who has created critical parts of the IP and is familiar with the architecture and code base. Alice also has the responsibility of encrypting the IP for customer delivery. Alice will be using the Questa SIM command line tools **vencrypt** (Verilog) and **vhencrypt** (VHDL). Alice uses a group of IEEE 1735 version 2 defined pragmas and command line switches (unique to Questa SIM) to create the encrypted files. Alice's main tasks include:

- Deciding on what methodology to follow in encrypting her IP. Alice has two choices discussed in the following section. One method is to use embedded source pragmas, and other method is to use "externalized recipes". This decision also implies what version of IEEE 1735 she will use.
- Deciding what is encrypted. This task includes Alice specifying what objects and files to expose to Rahul, who is Alice's customer.
- Deciding how text is encrypted. Alice creates "encryption recipes" directing how the command line tools process clear text source files. The IEEE 1735 protocol specifies various pragmas to accomplish this.
- Deciding on a targeted set of tools that can execute and process her encrypted source files. This requires Alice obtaining each targeted tool's public encryption key.
- Periodically Alice reviews and validates that Bob's public encryption keys are active. Bob as the tool vendor may need to update his public encryption key. Alice may need to re-encrypt her IP used by Rahul.
- Deciding on any special tool or encryption specific processes.

- *Rahul* is Alice's customer. He takes Alice's encrypted IP and uses it with Bob's tools. Rahul can only use tools targeted by Alice's encryption process. Rahul must obtain from Alice the list of supported vendor tools along with a listing of any objects, data structures, and functions he can access. Rahul does not need Bob's public encryption keys as the encryption tools embed that information in the encrypted files.
- *Bob* is a vendor whose tools Alice has targeted for her IP. Alice must get from Bob his public encryption keys. Bob's tools must follow the directions embedded in Alice's IP on how to decrypt the source to ensure interoperability. Alice expects Bob's tool to be reliable in this regard. That is, Bob must ensure that his tools can process IEEE 1735 encryption directives at the proper version level. This strictly includes following Alice's embedded pragmas on what data and objects may be exposed to Rahul. Bob's main interaction is with Rahul as Rahul uses Bob's tools directly.

Alice starts the encryption process with clear text source files such as the following "Hello world" Verilog module.

```
module top;
  initial
      $display("Hello world");
endmodule // top
```

IEEE 1735 Version Levels

The first decision Alice makes is deciding what version of the IEEE 1735 protocol she will use for encryption. The following table summarizes the version choices.

Table 3-1. IEEE 1735 Version Levels

IEEE 1735 Version	Notes	Example Encryption Commands
0	• Not interoperable. • No encryption envelopes. • Just the \`protect/\`endprotect transformed post-encryption to \`protected/\`endprotected • Older cryptography. • Recommended mostly for internal cross-team obfuscation.	Verilog source with v0 pragmas: **vencrypt top.v**
1	• The most widely implemented inter-operable encryption, as described in the language LRMs. • No rights management. • Some LRM features are not available (Questa SIM makes the version 2 viewport syntax available in version 1).	Verilog source with v1 pragmas **vencrypt top.v** Verilog source with no pragmas and a v1 header file **vencrypt -h recipe.v top.v**
2	• The latest interoperable encryption. • Adds rights and upgrades the cryptography.	Verilog source with v2 pragmas **vencrypt top.v** Verilog source with no pragmas **vencrypt -toolblock TOOL-KEY top.sv**

Alice chooses IEEE version 2. The Questa Sim encryption tools support this version with the exception of rights other than delegated. Alice checks to ensure that Bob's tools also support IEEE 1735 version 2.

Embedded Source Pragmas

Alice can use various combination of pragmas to achieve encryption. This section shows some of the options.

Embedded IEEE 1735 Version 2 Encryption Envelopes

While it is recommended that you use external encryption recipes, Alice can embed version 2 encryption envelopes into her source files.

Following is a “Hello World” example with an embedded IEEE 1735 version 2 encryption envelope. The presence of a toolblock implies this is a Version 2 encryption envelope.

```
//File IP2.v
// Must tell the encryption tool this is version 2
`pragma protect version = 2
// Empty common block
`pragma protect begin_commonblock
`pragma protect end_commonblock
// General information
`pragma protect data_method = "aes256-cbc"
`pragma protect author = "Questa Documentation"
`pragma protect author_info = "IEEE 1735 Version 2"
// Just one target tool in this example
`pragma protect begin_toolblock
`pragma protect key_keyowner = "Mentor Graphics Corporation"
`pragma protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`pragma protect key_method = "rsa"
`pragma protect rights_digest_method="sha256"
`pragma protect key_public_key
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtNA6tJ1tV/cXF4K5mL4s
4KCuTKWSbN/BnJJ6elRTWr2+s5Baaul0ctIX/3KYzpITmG9ph/4uZBs+jV5DAC+9
WRZQDc11JdIlRi04dEx/bGVbfPs3pdTPFZjA6gfegdW03ZNhjaJChTwEoXL1xIGP
oodJyhX9r1DoxU2lWB19vpwI5Geygh6pYgkPXb0aQzLh6hyUBhH9yMN6eV+imBbO
eax8ZCO6Gz2CJq3ebS/JoMYrikgcIEf6kVhIOiB9LluTp6TZlSd8ilwPhQmfXWH2
w4CaIpN8kADaVHnDWIdqqHlGf3cNQrlWj6FnFpSam6PjmWp5ZD4Jt6UNJxEoKEsn
gwIDAQAB
`pragma protect end_toolblock
`pragma protect begin
module IP;
   initial begin
      $display("Hello world");  end
endmodule // IP
`pragma protect end
```

Invoke the following command:

vencrypt IP2.v

This command produces the decryption envelope *IP2.vp*. as follows.

```
// Must tell the encryption tool this is version 2
// Empty common block
// Just one target tool in this example
`pragma protect begin_protected
`pragma protect version = 2
`pragma protect author = "Questa Documentation" , author_info = "IEEE 1735
Version 2"
`pragma protect encrypt_agent = "ModelSim" , encrypt_agent_info = "Questa
Sim 10.7b"
`pragma protect begin_commonblock
`pragma protect end_commonblock
`pragma protect begin_toolblock
`pragma protect key_keyowner = "Mentor Graphics Corporation" , key_keyname
= "MGC-VERIF-SIM-RSA-2"
`pragma protect key_method = "rsa"
`pragma protect encoding = ( enctype = "base64" )
`pragma protect key_block
McO0j1zM2+qc0yJSn4uMqZASvh6uzT9z+v3vYilj3uFW5Hk1dp4T40XP3UmuSDBt
C/EYXHp53KWr54Tl7QavDB4v6l27QJDsR+nJ565nbyvE7CjtP78LltvaLnp+U2uq
5iFsqt6cclLgzJYa52mRkuLJgo57cVASzR0iSr2oRxtUTUzkfwoTYD4tZgQFBazM
dHSbJcUsugnjFaaRYWJMKwNq5nJjql2IViLJgsl4AsIOeUfj5SK9HvzRwxjZJ7pY
giy9O1lFBJBiKAnyd2V+slQKEtt32/oAS12ZRXwjM1XeoFmkLY33sJJbwULND5Ms
jA8dDM5Q0d8qEzD9s5ogBA==
`pragma protect rights_digest_method = "sha256"
`pragma protect
end_toolblock="ylnY7xV1WJ0K9iXfpoJhKwhZ4+2f5XGIjXET3KmNpzU="
`pragma protect data_method = "aes256-cbc"
`pragma protect encoding = ( enctype = "base64" , line_length = 64 , bytes
= 176 )
`pragma protect data_block
uOnrawPNPWT05MoTp7PgdLhaS3kj/5MjyHp/Af/T8HdE7OdjEEXmRxSq88H6j+eH
yjo1+RR6HOu46sxsHMQ7utd5HDncKVhrePL0Zei2J/UPIl8rvkpf7m7P3HH3VtsK
eSBxDEmEKf301wy7ziEDzCyur7u5Tp6YsxOgDuSgFNB7ovGufbgPPvSvSLpK3REy
MmtbTLNw5J1daon6jU/U3U6yc397ro4b+Elt7Drjbs4=
`pragma protect end_protected
```

Embedded IEEE 1735 Version 1 Encryption Envelope

Alice can embed Version 1 encryption envelopes into IP source.

The following shows the “Hello World” example with an embedded encryption envelope following Version 1 methodology.

```
// File IP1.v
// Must tell the encryption tool this is version 1
`pragma protect version = 1
// General information
`pragma protect data_method = "aes256-cbc"
`pragma protect author = "Questa Documentation"
`pragma protect author_info = "IEEE 1735 Version 2"
// Just one target tool in this example
`pragma protect key_keyowner = "Mentor Graphics Corporation"
`pragma protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`pragma protect key_method = "rsa"
`pragma protect key_public_key
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtNA6tJ1tV/cXF4K5mL4s
4KCuTKWSbN/BnJJ6elRTWr2+s5Baaul0ctIX/3KYzpITmG9ph/4uZBs+jV5DAC+9
WRZQDc11JdIlRi04dEx/bGVbfPs3pdTPFZjA6gfegdW03ZNhjaJChTwEoXL1xIGP
oodJyhX9r1DoxU2lWB19vpwI5Geygh6pYgkPXb0aQzLh6hyUBhH9yMN6eV+imBbO
eax8ZCO6Gz2CJq3ebS/JoMYrikgcIEf6kVhIOiB9LluTp6TZlSd8ilwPhQmfXWH2
w4CaIpN8kADaVHnDWIdqqHlGf3cNQrlWj6FnFpSam6PjmWp5ZD4Jt6UNJxEoKEsn
gwIDAQAB
`pragma protect begin
module IP;
   initial begin
      $display("Hello world");
   end
endmodule // IP1.v
`pragma protect end
```

Invoke the command:

vencrypt VP1.v

This produces the decryption envelope *IP1.vp*.

```
// Must tell the encryption tool this is version 1
// General information
// Just one target tool in this example
`pragma protect begin_protected
`pragma protect version = 1
`pragma protect author = "Questa Documentation" , author_info = "IEEE 1735
Version 2"
`pragma protect encrypt_agent = "ModelSim" , encrypt_agent_info = "QA
Baseline Assertion: 10.8 Beta - 3305709"
`pragma protect key_keyowner = "Mentor Graphics Corporation" , key_keyname
= "MGC-VERIF-SIM-RSA-2"
`pragma protect key_method = "rsa"
`pragma protect encoding = ( enctype = "base64" , line_length = 64 , bytes
= 256 )
`pragma protect key_block
aTlJMq0JNfiOhqPb7jWz0snBAa1jYJ6oHLWdQzKZ+10ww7BlOnVgR5vESY8zsfYk
xL8NZVCqrbMu5Ewgt3g9ePLGKX7cV4DOy02Dsvkas2pzUD/v5ehFJ0M63cEAWUQS
+EWhVbrem3CZri/ysNScxSU3lpxrtuXBcLjUEEFhNVQlJxt6nlKwAlgEc3HRky6n
iuZTb/A9JWHO0pc3tM4Sfq7qSx49yqsL/68+YGuypGPj3U/EWmw5Uk2wolfYZJer
14FMjoVVCsIjB/+R5PL4pvnZp0cZiU4EBu50W0gm1s1aczC3bTu1QOi+2Ktv59UX
aeTsYfH+3VwRbtAtcwMDXQ==
`pragma protect data_method = "aes256-cbc"
`pragma protect encoding = ( enctype = "base64" , line_length = 64 , bytes
= 176 )
`pragma protect data_block
EBToeBkQp5sFqHNP/mK9w8gTpm5+wvDnx4MfFX6GZ5X/rfM1TuOC3ssWvYfYzgRm
0KcCOvYbUUgZriPPRHtqb8EZ77w1ArdGkvRby+te6Y3gY1hVn9CK8LwIElqBfPlh
/oLiYCkGM6gkOMBDDJE4iyq+eAwGFfiP86r3rFRUcJoGMGMcaGbLecQ3U9nvb5jP
8pZ52csD0CIn7smarVLIJLizmTmTMwijDhjwPvsH+Bk=`pragma protect end_protected
```

IEEE 1735 Version 0 Encoding

The Questa SIM tools **vencrypt** and **vhencrypt** support Version 0 encoding methods.

The following Hello World example uses Version 0 encoding.

```
// File IP0.V
`protect
module IP;
  initial begin
     $display("Hello world");
  end
endmodule // IP
`endprotect
```

Encrypt using following command:

```
vencrypt IP0.v
```

Command produces the file *IP0.vp*:

```
`protected
MTI!#OlJ~S[_GZ=O\e)ipmXme3#{+<=0Bx?<N1@vK|)[ypO'O)M-TXu+U!<2RGU}yn'\
[x{U?!U<nlK+<yio1^~{>AD?UDK+O-DWZ<i7_kx;~R)~=]-n1OEz7-s|Yk!m=HI]o\
*rLs?WB,Q[idk]kbhNRBIT}nGAB_B@oP/Gi\n~w,v
`endprotected
```

No Pragmas In Source IP

Alice can encrypt the “Hello World” example with no embedded pragmas.

The **vencrypt** and **vhencrypt** commands will encrypt an entire source file when it has no embedded pragmas. The commands infer that the entire file is to be encrypted, and to use the Questa SIM public encryption keys. The commands encrypt to Version 1.

```
module IP;
  initial begin
    $display("Hello world");
   end
endmodule // IP
```

Invoke the command:

vencrypt IP.v

The commands default to Version 1 producing the following decryption file *IP.vp.*

```
`pragma protect begin_protected
`pragma protect version = 1
`pragma protect encrypt_agent = "QuestaSim" , encrypt_agent_info = "QA
Baseline Assertion: 10.7b Beta - 3295444"
`pragma protect key_keyowner = "Mentor Graphics Corporation" , key_keyname
= "MGC-VERIF-SIM-RSA-2"
`pragma protect key_method = "rsa"
`pragma protect encoding = ( enctype = "base64" , line_length = 64 , bytes
= 256 )
`pragma protect key_block
SlbPp0nWpxc4Ff+tYRSay/BVCIcPIXArOln5UyUlLqdsBZlswflyYHKFN8xMbmTy
wJunBemyJrIffJvV1MISa1p9GRx1t+sBeUKIxRGBCjNCHLrf1ORJ834lPDAdEsi7
cDRgWbjVuYsvDIgLkVOx5rv5L9DJysnD4wdJ+M2zmZ6UUSw1BYr6a4Aotj/fppsA
FVJLf312vUyS77M8N3h3XmeKUMIWhDilqW3Yk18T5ti0c8B8pgiFNAZk/Cxt3063
rJfeTC3I/3PwdTh8VecqidP457NRuVSUJicB87ssc00+Egwua5ydBoo33InBtFkS
yFeDbS0oS19r0syFp6EGRg==
`pragma protect data_method = "aes128-cbc"
`pragma protect encoding = ( enctype = "base64" , line_length = 64 , bytes
= 176 )
`pragma protect data_block
xgEr0CyYMGR/XhFOjyU+MVF807KiMDTtQTXQMXm7o+oHVTgdAxRUDdpqAznYxf11
OxqbcIJwIahKvXn6KqOPqsyo+bMOpcNTqLFy/JV+iW6vhVkLbcfrRvo6KkaxXY22
gWwewYRc9PHf56mBwRFVcu8ze/1BNUmWe43XGY+b4SOiZigFK6wlksqkuH+gk8F6
2Lts0YmMeLZQaRVUvvuuAuoUKp+tnkCTSXQPJr8CH18=
`pragma protect end_protected
```

Externalized Encryption Recipes

Another set of options Alice can use as an encryption strategy is to take out encryption directives from source code.

Externalized IEEE 1735 Version 1 Encryption Recipe

If Alice uses IEEE Version 1, she can still implement externalized encryption recipes.

The following “Hello World” example only contains begin and end pragmas. There are no encryption envelopes in the source. When the encryption commands do not find any encryption envelope directives, then the Questa SIM tools default to using the Questa SIM public encryption key recipe located in the *keyring* directory. The commands set the version level to 1. The name of the encryption recipe is *MGC-VERIF-SIM-RSA-2.active*

```
// File IP_2v
`pragma protect begin
module IP;
  initial begin
     $display("Hello world");
  end
endmodule // IP
`pragma protect end
```

Enter the command:

vencrypt IP_2.v

This produces the decryption envelope *IP_2.vp.*

```
`pragma protect begin_protected
`pragma protect version = 1
`pragma protect encrypt_agent = "ModelSim" , encrypt_agent_info = "QA
Baseline Assertion: 10.8 Beta - 3305709"
`pragma protect key_keyowner = "Mentor Graphics Corporation" , key_keyname
= "MGC-VERIF-SIM-RSA-2"
`pragma protect key_method = "rsa"
`pragma protect encoding= ( enctype = "base64" , line_length = 64 , bytes
= 256 )
`pragma protect key_block
EpsMsTzGLgV6LcdFMccuOKIOd+iW2zilZk/Dx+iBwPO0LArGcAlsMbqi4Uo4CuOK
+rzuNFAbVoluO5PhI4tq1sKffWo/ougZLjAo6ZX3AoGqJ8hNkQrjwg/TF0MZfYAA
+wyPZTVIOCfeoj79LzHrrSFHXqSQKYQzw7DmUF9ez+YvVR/IvPVw6KHsfiVFynne
JG9R1hnycmumLfaDwZd34hg+F8ZeH3VhPq/NDxU6hOwFAWRhLWwGFeWFXHgQzLu5
adz0NVBUcigSJO3CYqEuO4bL6y0w7CAN82cHW7YMTnRIEZl40d9/bWNbF+NMiogO
rYqzBoeLWhsyOg0/Qz8gaw==
`pragma protect data_method = "aes128-cbc"
`pragma protect encoding = ( enctype = "base64" , line_length = 64 , bytes
= 176 )
`pragma protect data_block
EDmsAWhUB2pIUk7JjcCFCVuWwykYwZ3c3jACmk/UWICDZvRw1PNxP/TawYYiAOc1
R2eRZl63kLgSV5ib9h9hrEFqUFYps4f1CoLOoYhicBZKBGafAu5JpVBzOFt6R3W1
hOpR2ljEaBJ1aPed+Sk1YgbYi/b4XHXWNzWUN0GIiDWxrbpzbcTzF9mVGoApu2AW
a+RXKMJWgdSpj6lruVAPshANp8X/6Zf5nTt7xqL4VJ4=
`pragma protect end_protected
```

IEEE 1735 Version 1 External Header Recipe Files

One method Alice can use to implement externalized encryption recipes is through the use of header files containing the encryption recipes.

The following file *h1.v* is a header file that contains pragmas defining the Questa SIM public encryption key. However, it can contain any encryption recipe.

```
//h1.v
// Included as a header file.
`pragma protect data_method = "aes256-cbc"
`pragma protect author = "Questa Documentation"
`pragma protect author_info = "IEEE 1735 Version 2"
`pragma protect key_keyowner = "Mentor Graphics Corporation"
`pragma protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`pragma protect key_method = "rsa"
`pragma protect key_public_key
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtNA6tJ1tV/cXF4K5mL4s
4KCuTKWSbN/BnJJ6elRTWr2+s5Baaul0ctIX/3KYzpITmG9ph/4uZBs+jV5DAC+9
WRZQDc11JdIlRi04dEx/bGVbfPs3pdTPFZjA6gfegdW03ZNhjaJChTwEoXL1xIGP
oodJyhX9r1DoxU2lWB19vpwI5Geygh6pYgkPXb0aQzLh6hyUBhH9yMN6eV+imBbO
eax8ZCO6Gz2CJq3ebS/JoMYrikgcIEf6kVhIOiB9LluTp6TZlSd8ilwPhQmfXWH2
w4CaIpN8kADaVHnDWIdqqHlGf3cNQrlWj6FnFpSam6PjmWp5ZD4Jt6UNJxEoKEsn
gwIDAQAB
```

The following command encrypts the source files using external encryption recipes, but as an included header file. The file *IP1_2.v* is the “Hello World” example with only **begin** and **end** pragmas. The -h switch specifies a header file to include with the “Hello World” example. The -o switch redirects output to the named file.

```
vencrypt -header=h1.v -o IP1ext1.vp IP1_2.v
Reading IP1_2.v
Encrypting (1) into IP1ext1.vp with header h1.v.
Creating IP1ext1.vp
Errors: 0, Warnings: 0
```

The command produces the following decryption IEEE 1735 Version 1 envelope file *IPext1.vp*:

```
`pragma protect begin_protected
`pragma protect version = 1
`pragma protect author = "Questa Documentation" , author_info = "IEEE 1735
Version 2"
`pragma protect encrypt_agent = "QuestaSim" , encrypt_agent_info = "QA
Baseline Assertion: 10.7b Beta - 3295444"
`pragma protect key_keyowner = "Mentor Graphics Corporation" , key_keyname
= "MGC-VERIF-SIM-RSA-2"
`pragma protect key_method = "rsa"
`pragma protect encoding = ( enctype = "base64" , line_length = 64 , bytes
= 256 )
`pragma protect key_block
r5CZUBwtMi0BDJ/tH1UT+cYlIQ6jtXBQHwNH/pAuVAzGoA3TrcSiV7zd7RJPoEc4
CbBvtz9hbvWt56W1kjco2pxLiZZoa8OG/PuNzstIxExvNj9VMWuCBD6nIYcLujeZ
P4uYT2Hw/HHEL4lWoAq3tYntKZKjcJELJv1vMjBPcHBvoBVyvzHBl7Bu3hDC0DaK
40EqF+iDQH+RnDIQh6NBepOpTsEhKoNY0XEzFC2gfkdPJceIV69xTuynXY9UiypW
fTRXluEuTJOl+7RFTRJsPVQjD+AUcmGETsTQ0N3YX5GRr0zsZNRWq3tUeOwcnw4z
9AX833N1P3yyj51OzyXyrQ==
`pragma protect data_method = "aes256-cbc"
`pragma protect encoding = ( enctype = "base64" , line_length = 64 , bytes
= 176 )
`pragma protect data_block
A9p06jRcthNgBnZNTmfFf66MUF+V7qP/r/2nN7FmxixxhN7ti6TSoc3rv5gWSa2U
hIX1yDNCssH3KTZB89DUBtbX0h/ULCnHEoc0k/GJDkoWETc+Ksg9A5J2H9rV0zH0
/PLmJm26ox9AdhmVH6vHkb3BaYuS1nBp1Pw2XaI5ZMv4BwALzlczZn7X8O1PVvj/
f9LWsxmvejrRMZIfuQD5xygczhsEiNXItHkUChT4bhw=
`pragma protect end_protected
```

Decryption Envelopes

From this point Alice will use IEEE 1735 specified pragmas, Bob’s public encryption keys, and various command line arguments to produce the final output of the encryption process which is a *decryption envelope*. Alice gives Rahul a final set of files with decryption envelopes. Rahul uses these files as input into Bob’s tools. Bob’s tools use his private encryption key to decrypt the IP.

The contents of the source is not important at this point, only the output decryption envelope. By understanding the decryption envelope, Bob, Alice and Rahul will understand the inputs required to produce an encrypted file.

```
`pragma protect begin_protected
`pragma protect version=2
`pragma protect encrypt_agent="Mentor Graphics, Inc.", encrypt_agent_info="Questa 10.7b"

`pragma protect  begin_commonblock
`pragma protect  end_commonblock

`pragma protect  begin_toolblock
`pragma protect key_keyowner="ABC Tools"
 `pragma protect key_keyname="ABC-KEY-1024"
 `pragma protect key_method="rsa"
 `pragma protect rights_digest_method="sha256"
 `pragma protect key_block
LVNUh72QqU4giJYSCZXnwfBRqAhUgKadk8P5C4FcQYB8Amf+y5GpzE1vf1xog/Xr
+gBBCxJ2KLvYWufLjM6ez3/9GTJOgGpRijQUFu9nhqFCtWb5zJOWhRMqau1tFszZ
eu906q+8n/lWtCbdYmq8DzIL1Vpunf/SXOrbPNJN2lM=

 `pragma protect end_toolblock="b/7dpb9vQtHMBxsHFWHVOXL07X6oEtkkX5ys="
 `pragma protect  begin_toolblock
 `pragma protect key_keyowner="Mentor Graphics Corporation"
 `pragma protect key_keyname="MGC-VERIF-SIM-RSA-2"
 `pragma protect key_method="rsa"
 `pragma protect rights_digest_method="sha256"
 `pragma protect key_block
KyCByJ5KXJSmGKYSRxhmF091VakDdm1IoiZQ4ta4UrZlr90OaOasmZAdDWEhXmpj
Pl7s3ubwt4QNbL1HKJpmsvlxyThWxgXjN3VtQcTDHNA3Q554HH2GZeOEQO6L/JU/
alwloeJKeUZD0uLRaDpJKE/ep90+W3wr1kW6pVCNguhBsDrzCy/0vkbPTqA5Ya23
T1zCoLvgreCMFkjD43uAFXao3J7lNeOFSKouxDBsSk2P/SiLFBQ9+Na/4YowZTSR
NaZAuWY/Bp3tsA7d5D4pTvwezhpBSBrzECJtlAT1j7LgZcMQI1FZW9waJtRuHcIS
G+Pwgrl0sUY4Hsb1gtw+fw==

 `pragma protect end_toolblock="C0XGjmTI1AO4Wg61NLmo2j3Ng1I84FM5IYiz="
 `pragma protect data_method="aes128-cbc"
 `pragma protect data_block
ZV/gs4OIhQrYmUjl78J0//LyJcAeuhHuUoj11DDwFW4=
 `pragma protect end_protected
```

There are three important sections in the decryption envelope that allow Bob's tools to properly decrypt the source:

- Bob's tools must find a public key that they recognize to use the decrypted code. The owner of the key is specified by the **key_keyowner** and **key_keyname** pragmas. This gives Bob's tools everything they need to start the decryption.
- Bob's tools use their recognized key to get the secondary session key to the data. This key is used to decrypt the **data_block**. The **data_method** pragma tells Bob exactly how to use the key.The encrypting tools communicates the session key to the decrypting tool, by means of a **KEY_BLOCK**. The decrypting tool reads each **KEY_BLOCK** until it finds one that specifies a key it knows about. It then decrypts the associated **KEY_BLOCK** data to determine the original session key, and uses that session key to decrypt the IP source code.
- Encrypted data. These are marked off by the **data_block** pragma.

At various points in the encryption process, Alice must specify all the previous information to the **vencrypt** and **vhencrypt** commands.

The following is a walk-through of the previous decryption envelope line-by-line which reflects how Alice produced them for Rahul and Bob.

- *Line 1.* The result of a **`pragma protect begin** pragma in the source text, or, it was inferred in a whole-file encryption. The smame could also have been achieved by Alicia using the using also the command line argument –wholefile.
- *Line 2.* The IEEE 1735 version number. May be explicit in the header file (-h) or specified with embedded pragmas.
- *Line 3.* Inserted by the encryption tool to identify itself.
- *Line 5-6.* (Version IEEE 1735 v2 only). The common block is replicated from either the embedded text, the -h header file argument, or the -common file argument. It may contain common rights (this example does not).
- *Lines 8-18 and 19-32.* There are two toolblocks, each one for a specific tool. These may have come from two separate command line options: the -toolblock option, from within a -h header file; or embedded pragmas. If Alice does not specify a toolblock, then the encryption commands infer and default to the Questa Sim public encryption key.
- *Line 33.* Specified from the command line, an embedded pragma, or in the files specified by the -common or -h command lines arguments. The encryption commands infer the value of data_method if Alice does not specify it. The inferred value is "aes128-cbc".
- *Lines 34-36.* The result of the input data after the original **`pragma protect begin** and up to the **`pragma protect end** (or EOF if inferred).

Source Code Pragmas

Alice relies on the **viewport** and **interface_viewport** pragmas to expose objects as the primary method within source files. The remaining source is inferred to be hidden by the **vencrypt** and **vhencrypt** commands.

In Alice's source files, there will be only four pragmas for either Verilog or VHDL listed below. If Alice does not insert any visibility pragmas into her source code, then the encryption tools process and encrypt the entire file for both Verilog and VHDL.

- Verilog
    - `pragma protect begin
    - `pragma protect end
    - `pragma protect interface_viewport

    - `pragma protect viewport
- VHDL
    - `protect begin
    - `protect end
    - `protect interface_viewport
    - `protect viewport

Chapter 4 Autoprotect Mode

The **vencrypt** command supports pragma-less encryption that only supports IEEE 1735 version 0. This feature is called Autoprotect Mode. It only supports the encryption of Verilog source files.

Autoprotect Mode is meant for in-house obfuscation methodologies for such tasks as black box testing. It is not recommended that you use this feature for encrypting IP for external use.

Automatic Encryption Operation

Automatic encryption allows you to encrypt Verilog/SystemVerilog files without the use of pragmas and other directives. In addition, Autoprotect Mode supports PLI access into protected modules.

Note

Mentor Graphics recommends using automatic encryption mode only for internal uses such as black box testing.

Autoprotect Mode Process and Limitations

- The **vencrypt** command protects the text between "module <name>" and "endmodule" statements. There is one option to encrypt an entire file regardless of module boundaries.
- For most arguments, the **vencrypt** command does not encrypt text outside a module.
- Automatic encryption does not require any embedded protect directives or the specification of any encryption key.
- The **vencrypt** command encloses all encrypted text with the `protected and `endprotected directives.
- The **vencrypt** command in Autoprotect Mode ignores any embedded protect directives. The command replaces directives with spaces in the encrypted output file.
- Various arguments allow you to control what is visible within a module. Refer to Table 4-1 for a list of the arguments.
- There is an argument which provides PLI users access to encrypted modules.
- Autoprotect Mode produces an encrypted file with a **.vp* extension.

You can use only one of the Autoprotect Mode arguments for any particular file.

Table 4-1. Autoprotect Mode Arguments

Argument	Module Name Visibility	Port List Visibility	Parameter List Visibility
autoprotect	Yes	No	No
autoprotect_all	No	No	No
auto2protect	Yes	Yes	No
auto3protect	Yes	Yes	Yes
auto5protect *	Yes	Yes	Yes
* Encrypts everything in between `ifdef/‘endif.			

You cannot use Autoprotect Mode with the following **vencrypt** arguments:

- allow_voloce
- wholefile
- data_method
- common
- toolblock
- keyring
- header

PLI Support

To support PLI access, the vencrypt command has a -pli_protected argument. You can use this argument in conjunction with any one of the Autoprotect Mode arguments. Only modules encrypted with the -pli_unprotected argument allow access.

```
vencrypt -pli_unprotected -autoprotect my_file.v
```

Using Autoprotect Mode

Use Autoprotect Mode to encrypt Verilog/SystemVerilog source without the use of pragmas.

Prerequisites

- Source files restricted to Verilog.

Procedure

1. Review and decide what elements of a module you want to encrypt or make visible. Refer to Table 4-1 on page 72 for a list of options.
2. Decide if a PLI program should access the encrypted code. If yes, then you must use the -pli_unprotected argument in conjunction with an Autoprotect Mode option.
3. Create a Verilog source file. The following is a sample file to demonstrate automatic encryption.

```
module test_dff4(output [3:0] q, output err);
parameter WIDTH = 4;
parameter DEBUG = 0;
   reg [3:0] d;
   reg   clk;
   dff4 d4(q, clk, d);
assign
  err = 0;initial
   begin
      $dump_all_vpi;
      $dump_tree_vpi(test_dff4);
      $dump_tree_vpi(test_dff4.d4);
      $dump_tree_vpi("test_dff4");
endendmodule

module dff4(output [3:0] q, input clk, input [3:0] d);
`pragma protect data_method = "aes128-cbc"
`pragma protect author = "IP Provider"
`pragma protect author_info = "Widget 5 version 3.2"
`pragma protect key_keyowner = "Mentor Graphics Corporation"
`pragma protect key_method = "rsa"
`pragma protect key_keyname = "MGC-VERIF-SIM-RSA-2"
`pragma protect begin
   dff_gate d0(q[0], clk, d[0]);
   dff_gate d1(q[1], clk, d[1]);
   dff_gate d2(q[2], clk, d[2]);
   dff_gate d3(q[3], clk, d[3]);
endmodule // dff4

module dff_gate(output q, input clk, input d);

wire preset = 1;
wire clear = 1;
nand #5
      g1(l1,preset,l4,l2),
      g2(l2,l1,clear,clk),
      g3(l3,l2,clk,l4),
      g4(l4,l3,clear,d),
      g5(q,preset,l2,qbar),
      g6(qbar,q,clear,l3);
endmodule
wire test =1;
`pragma protect end
```

4. Create encrypted source using the **vencrypt** command, such as:

   vencrypt -auto2protect myfile.v

   The resulting *myfile.vp* follows. Note that the encrypted text is formatted to the width of printed page. The resulting encryption the name of the module as clear text. Non-automatic encryption encrypts all visible objects in the source file.

```
module test_dff4(output [3:0] q, output err);

'protected
MTI!#\!p3asr=j=+aiG#kG!YelA,T[
Tn![{$""F^|]@WBNj3{u}>X5{=+',C2VB>$mE@lODF}JDi{{z<0q^_=Y^x7s}xkUlY[
p?1$vRjU7]+j]yK${p\
Z~[K'+nojx[#1p<=@9x?*Z6^g@]ZIV_xig73ovC!zW$I$IIt>UXRH-
KRRD+]xpw~ur>#oCo,3j^r]+_@6>{BW?EDx^m-w+[VUoe2rf3s+Xl?xEJ17p-
eAEnE;#k{;]x@Tn53'lL][#nBC$ZVZI-B2[,05e1nm1BE*jR@lih7H'B*/
BIRo>sY#oO?OpOOYrxk,$Z\UZrqF~-3-^W2T!5vm}aE13*m'Woa=ar+R"$K'EGw]RI7
`endprotected
endmodule

module dff4(output [3:0] q, input clk, input [3:0] d);

`protected
MTI!#w1,^RHum@}eI5vujVK>X:;G{#@-TY:u["F^|]"W$*3-
rKmO{uUY?ji]k1eunCl@N%:#RQ,\*w]j<sek>zw]$Cj-
w[[5OjR;*AGHD+oj5@[UnsKOY?$VBIAz_V'lz\#$o+sao?$1$Kvka}-
`endprotected
endmodule //

dff4module dff_gate(output q, input clk, input d);

`protected
MTI!#5^7_y$7ekr3@ZWB\u'r7z;G{#@-
TY:j["F^|]"Ru*5{CwlR*iwY?VA]5EQ+Xl=$Y{s_wQ-j,r_qYTRu'#HK-
[?#5EH[Kv{JY(I*B\Tw-xuXY~2[z}C3C2=;\21C-
xd$BA;O?~r125Y(A]Z?j3v{v'Ya_Ul\EpkBlH,VQxBOox<T-
G!Q}3m'7~vQ,j<IvT$}]AeR$elzRo~~3$7s^#K2[HQRk$AZOaOAGU[$NRuEX3n\$_i
`endprotected
endmodule
wire test =1;
```

End-User License Agreement with EDA Software Supplemental Terms