

CDN P1364e™/D1

May 29, 2003

# **Draft Standard for Verilog Protected Envelopes**

Copyright © 2003 by Cadence Design Systems, Inc.

This document is an unapproved draft of a proposed IEEE Standard. As such, this document is subject to change. **USE AT YOUR OWN RISK!** Because this is an unapproved draft, this document must not be utilized for any conformance/compliance purposes.

## Contents

Contents	2
2. Lexical Conventions	4
2.9 Pragmas	4
19. Compiler Directives	4
19.10 Protected Envelopes	4
19.10.1 Processing protected envelopes	5
28. Protected Envelopes	6
28.1 Envelope Directives	6
28.2 Envelope encoding keywords	7
28.2.1 begin	7
28.2.2 end	8
28.2.3 begin_protected	8
28.2.4 end_protected	8
28.2.5 author	9
28.2.6 author_info	9
28.2.7 encrypt_agent	9
28.2.8 encrypt_agent_info	10
28.2.9 encoding	10
28.2.10 data_keyowner	11
28.2.11 data_method	11
28.2.12 data_keyname	12
28.2.13 data_public_key	13
28.2.14 data_decrypt_key	13
28.2.15 data_block	13
28.2.16 digest_method	14

28.2.17 digest_block	14
28.2.18 key_keyowner	15
28.2.19 key_method	15
28.2.20 key_keyname	15
28.2.21 key_public_key	16
28.2.22 key_block	16
28.2.23 decrypt_license	17
28.2.24 runtime_license	17
28.2.25 comment	18
28.2.26 reset	18
28.2.27 viewport	19
A.1. Encryption/Decryption Flow	20
A.1.1 Tool Vendor Secret key encryption system	20
A.1.1.1 Encryption Input	20
A.1.1.2 Encryption output	20
A.1.2 IP Author secret key encryption system	21
A.1.2.1 Encryption Input	21
A.1.2.2 Encryption output	21
A.1.3 Digital Envelopes	22
A.1.3.1 Encryption Input	22
A.1.3.2 Encryption output	23

# Amendment for Verilog Protected Envelopes

## 2. Lexical Conventions

Editor's Note: Add the following sub-clause at the end of this clause as 2.9:

### 2.9 Pragmas

A pragma is a mechanism for annotating the Verilog HDL with a structured specification that alters interpretation of the Verilog source in an implementation-specified way. This sub clause specifies the syntactic mechanism that shall be used for specifying pragmas, without standardizing on any particular pragmas.

pragma ::= (From Annex A - A.???)

```
/* pragma pragma_name { pragma_expression } ; */
```

```
// pragma pragma_name { pragma_expression } ; \n
```

pragma\_expression ::= pragma\_keyword

```
| pragma_keyword = pragma_value
```

```
| pragma_value
```

pragma\_value ::= constant\_expression

```
| string
```

pragma\_keyword ::= identifier

## 19. Compiler Directives

Editor's Note: Add the following sub-clause at the end of this clause as 19.10:

### 19.10 Protected Envelopes

Protected Envelopes specify a region of text which shall be transformed prior to analysis by the source language processor. These regions of text are structured to provide the source language processor with the specification of the cryptographic algorithm, key, envelope attributes, and textual design data.

All information which identifies a Protected Envelope is introduced by the **protect** pragma. This pragma is reserved by this standard for the description of Protected Envelopes, and is the prefix for specifying the regions and processing specifications for each protected envelope. Additional information is associated

with the pragma by appending pragma expressions. Consecutively specified pragma expressions may be concatenated in a white space separated list as a single protect pragma declaration, or may be specified in individual protect pragma declarations.

Envelopes may be defined for either of two modes of processing. Encryption envelopes specify the pragma expressions for encrypting source text regions. Decryption envelopes specify the pragma expressions for decrypting encrypted text regions. Decryption envelopes may contain other envelopes within their enclosed data block. The number of nested decryption envelopes that can be processed is implementation-specified.

```
hdl_envelope ::=      encrypt_envelope
                    | decrypt_envelope

encrypt_envelope ::= envelope_params  protect_begin_pragma  content_params  source_text
protect_end_pragma

decrypt_envelope ::= envelope_params  protect_begin_protected_pragma  content_params
encrypted_text protect_end_protected_pragma

envelope_params ::= { protect_keyword_pragma }

content_pragmas ::= { protect_keyword_pragma }
```

Note:

source\_text: The source text encompasses all the text, comments, included pragma directives, user code etc.

string: Double quotes (“”) can be used around string values that contain white space characters, but are optional if no embedded white space is present. If present, double quotes are stripped from the actual value of the pragma expression. If double quote characters occur in a pragma expression value, then they must be escaped with \ (e.g. \).

encrypted\_text: is the encrypted binary data encoded in printable characters, spanning over multiple lines. The current active encoding scheme is used, that specifies the algorithm (uuencode/ sixel), the line width on each line etc.

Pragma expressions that precede **begin** or **begin\_protected** are designated as envelope keywords. Those pragma expressions which follow the **begin/begin\_protected** keywords and precede the associated **end/end\_protected** keywords are designated as content keywords. Content keywords are those pragma expressions which are within the region of text that is processed during encryption or decryption of a protected envelope.

Adjacent protect pragmas with no intervening source text are equivalent to a single protect pragma whose pragma expressions are the ordered concatenated list of pragma expressions from those adjacent pragmas.

### 19.10.1 Processing protected envelopes

Two modes of processing are defined for protected envelopes. Envelope encryption is the process of recognizing encryption envelopes in the source text and transforming them into decryption envelopes. Envelope decryption is the process of recognizing decryption envelopes in the input text and transforming them into the corresponding clear text for the parsing step that follows.

### 19.10.1.1 Encryption

Verilog tools that provide encryption services shall transform source text containing encryption envelopes by replacing each encryption envelop with a decryption envelope formed by encrypting the source text of the encryption envelope according to the specified pragma expressions.

Source text which is not contained in an encryption envelope shall not be modified by the encrypting language processor.

Decryption envelopes are formed from encryption envelopes by transforming the specified encryption envelope pragma expressions into decryption envelope pragma expressions and decryption content pragma expressions. A session key is created, and a copy encrypted by the exchange key is recorded in the decryption envelope. Next the body of the encryption envelope is encrypted using the session key and is recorded in the decryption envelope as a **data\_block**.

### 19.10.1.2 Decryption

Verilog tools that support decrypting compilation shall transform source text containing decryption envelopes by replacing each encryption envelop with the decrypted form of envelope formed by encrypting the source text of the encryption envelope according to the specified pragma expressions. This substitution shall occur in a manner similar to and at a translation phase consistent with that of macro substitution.

## 28. Protected Envelopes

Editor' s Note: Add this clause to the end of the normative content as clause 28:

### 28.1 Envelope Directives

Protected envelopes are specified as lexical regions delimited by protect pragma declarations. The semantics of a particular protect pragma declaration is specified by its pragma expressions. This standard reserves the keyword names listed in the following table for use as keywords to the protect pragma. These keywords are defined in clause 28.2, with a specification of how each participates in the encryption and decryption processing modes.

begin	Opens a new encryption envelope
end	Closes an encryption envelope
begin_protected	Opens a new decryption envelope
end_protected	Closes a decryption envelope
author	Specifies the author of an envelope
author_info	Specifies additional information about the author
encrypt_agent	Identifies the encryption service
encoding	Specifies the coding scheme for encrypted data
data_keyowner	Identifies the owner of the data encryption key
data_method	Identifies the data encryption algorithm
data_keyname	Specifies the name of the data encryption key
data_public_key	Specifies the public key for data encryption
data_decrypt_key	Specifies the session key for data encryption
data_block	Begins an encoded block of encrypted data
digest_method	Specifies the digest computation algorithm
digest_block	Specifies an authentication code for data integrity

key_keyowner	Identifies the owner of the key encryption key
key_method	Specifies the key encryption algorithm
key_keyname	Specifies the name of the key encryption key
key_public_key	Specifies the public key for key encryption
key_block	Begins an encoded block of key data
decrypt_license	Specifies licensing constraints on decryption
runtime_license	Specifies licensing constraints on simulation
comment	Uninterpreted documentation string
reset	Resets pragma keyword values to default
viewport	Modifies scope of access into protected envelope

The scope of **protect** pragma declarations is completely lexical and not associated with any declarative region or declaration in the HDL text itself. This lexical scope may cross file boundaries, included files etc.

In the protection envelopes where a specific pragma keyword is absent, the Verilog tool shall use the default value. Verilog tools that perform encryption should explicitly output all relevant pragma keywords for each envelope in order to avoid unintended interpretations during decryption. Further robustness can be achieved by appending a **reset** pragma keyword after each envelope.

## 28.2 Envelope encoding keywords

### 28.2.1 begin

#### 28.2.1.1 Syntax

`begin`

#### 28.2.1.2 Description

ENCRYPTION INPUT: The **begin** pragma expression is used in the input text to indicate to an encrypting tool the point at which encryption should begin. All text, including comments and other protect pragmas, between the **begin** pragma expression and the corresponding **end** pragma expression should be encrypted and are stored in the output format using the **data\_block** pragma expression.

Nesting of pragma **begin/end** blocks is not supported, although there may be **begin\_protected/end\_protected** blocks containing previously encrypted content inside such a block. They are simply treated as a byte stream and encrypted as if they were text.

An implementation should not actually encrypt the pragma **begin** because the existence of known clear text at the beginning of a message can weaken many algorithms.

In many cases additional pragmas will follow the **begin** keyword and therefore be included in the encrypted output. Comments are explicitly included in the encrypted region because increased security can be achieved by including a random length, randomly generated set of comments in the encrypted region. Without this caveat, algorithms which attack known clear text looking for Verilog or pragma keywords at the start of an encrypted block could be used.

ENCRYPTION OUTPUT: none

DECRYPTION INPUT: none

## 28.2.2 end

### 28.2.2.1 Syntax

end

### 28.2.2 Description

ENCRYPTION INPUT: The **end** pragma expression is used in the input clear text to indicate the end of the region that should be encrypted. Note that an implementation should not actually encrypt the pragma **end** because the existence of known clear text at the end of a message can weaken many algorithms.

ENCRYPTION OUTPUT: none

DECRYPTION INPUT: none

## 28.2.3 begin\_protected

### 28.2.3.1 Syntax

begin\_protected

### 28.2.3.2 Description

ENCRYPTION INPUT: If found in an input file during encryption **begin\_protected/end\_protected** block and its contents should be treated as input clear text. This could result from a situation where a previously encrypted model is being re-encrypted as a portion of a larger model. An additional requirement is that any other protect pragmas inside the **begin\_protected/end\_protected** block should not be interpreted or override pragmas in effect. In this way, nested encryption will not corrupt pragma values in the current encryption in process.

ENCRYPTION OUTPUT: After encrypting a **begin/end** block during encryption, the encrypting tool should produce a corresponding **begin\_protected/end\_protected** block in the output file. This block begins with the **begin\_protected** pragma expression. Following **begin\_protected** all pragma expressions required as encryption output should be generated prior to outputting the **end\_protected** pragma expression. In this way protected blocks are completely self-contained avoiding any undesired interaction when using multiple encrypted models during the decryption process.

Note that this does not begin a block of encrypted data or keys, the **data\_block** and **key\_block** pragma expressions are used for this purpose and they will always be found within a **begin\_protected/end\_protected** block.

DECRYPTION INPUT: The **begin\_protected** pragma expression begins a previously encrypted region. A decrypting tool should accumulate all the pragma expressions in the block for use in decryption of the block.

## 28.2.4 end\_protected

### 28.2.4.1 Syntax

end\_protected

### 28.2.4.2 Description

ENCRYPTION INPUT: This pragma expression indicates the end of a previous **begin\_protected** block. This indicates that the block is complete and new pragma expression values will be accumulated for the next envelope.

ENCRYPTION OUTPUT: The **end\_protected** pragma expression should be output to indicate the end of a protected block.

DECRYPTION INPUT: The **end\_protected** pragma expression indicates the end of a set of pragmas that should be sufficient to decrypt the current block. Upon encountering **end\_protected** a tool should verify that all required information is present.

### 28.2.5 author

#### 28.2.5.1 Syntax

```
author=<string>
```

#### 28.2.5.2 Description

ENCRYPTION INPUT: The **author** pragma expression is used to indicate the name of the IP author. It should be given outside any begin/end block so that this information is transferred to clear text in the output file.

ENCRYPTION OUTPUT: The **author** pragma expression should be output in each protected block unchanged from the input.

DECRYPTION INPUT: none.

### 28.2.6 author\_info

#### 28.2.6.1 Syntax

```
author_info=<string>
```

#### 28.2.6.2 Description

ENCRYPTION INPUT: The **author\_info** pragma expression is provided to allow arbitrary information to be provided by the IP Author in the form of a string value. Its use is strictly optional and the contents are not required in any way during encryption or decryption.

ENCRYPTION OUTPUT: The **author\_info** pragma expression should be output in each protected block unchanged from the input.

DECRYPTION INPUT: none

### 28.2.7 encrypt\_agent

#### 28.2.7.1 Syntax

```
encrypt_agent=<string>
```

### 28.2.7.2 Description

ENCRYPTION INPUT: none

ENCRYPTION OUTPUT: The **encrypt\_agent** pragma expression should be output as clear text in each protected block. It takes a string value indicating the name of the encrypting tool. This is the tool vendor or tool being used to perform the encryption. This key-word is optional in all cases but may be included to document the toolset performing the encryption.

DECRYPTION INPUT: none

### 28.2.8 encrypt\_agent\_info

#### 28.2.8.1 Syntax

encrypt\_agent\_info=<string>

#### 28.2.8.2 Description

ENCRYPTION INPUT: none

ENCRYPTION OUTPUT: The **encrypt\_agent\_info** pragma expression is provided to allow arbitrary information to be provided by the encrypting tool in the form of a string value. Its use is strictly optional and the content is not required in any way during encryption or decryption.

DECRYPTION INPUT: none

### 28.2.9 encoding

#### 28.2.9.1 Syntax

encoding=<encoding descriptor>

#### 28.2.9.2 Description

ENCRYPTION INPUT: The **encoding** pragma expression specifies how pragma expressions and encrypted text shall be encoded. The encoding is necessary to ensure that this potentially binary data can be re-inserted into a text document without impairing the subsequent editing or transmission of the document. If an **encoding** pragma expression is present in the input stream it specifies how the output should be encoded.

The following sub-keywords values are specified for the value of the <encoding descriptor> of the **encoding** pragma expression. Each of them are found in the pragma expression string value given as the <encoding descriptor> and are separated by white space.

**encoding**:<encoding\_type> - specifies the method for calculating the encoding.

raw	identity transformation
uuencode	method specified in IEEE 1003.1-2001 (uuencode Historical Algorithm)
base64	method specified in IETF RFC 2045 (also IEEE 1003.1-2001 uuencode -m)
quoted-printable	method specified in IETF RFC 2045

If **raw** then no encoding has been performed and the encoded data may contain non-printable characters. Further encoding methods may be added in the future.

**line\_length**:<number> - this is the number of characters (after any encoding) in a line of the **data\_block**. This allows the insertion of line breaks in the **data\_block** after encryption and encoding to make embedding in ASCII formats simpler. Without the additional line breaks the **data\_block** would typically exceed the line length requirements of commonly used editors (such as vi) and make the containing file not editable.

**bytes**:<number> - this is the number of bytes in the original block of data before any encoding or the addition of line breaks.

ENCRYPTION OUTPUT: The **encoding** directive should be output in each **begin\_protected/end\_protected** block to explicitly specify the encoding used by the **encrypt\_agent**. A tool may choose to encode the data even if no **encoding** pragma expression was found in the input stream and should output the corresponding **encoding** pragma expression.

The **data\_block**, **data\_public\_key**, **data\_decrypt\_key**, **digest\_block**, **key\_block**, and **key\_public\_key** are all encoded using this encoding. If separate encoding is desired for each of these fields then multiple **encoding** pragma expressions can be given in the input stream prior to each of the above pragma expressions. The **bytes** value is added by the encrypting tool for each block that it encrypts.

DECRYPTION INPUT: During decryption, the **encoding** directive is used to find the encoding algorithm used and the size of actual data. The decoded data is then used for further processing.

## 28.2.10 data\_keyowner

### 28.2.10.1 Syntax

`data_keyowner=<string>`

### 28.2.10.2 Description

ENCRYPTION INPUT: The **data\_keyowner** specifies the company or tool that is providing the keys used for encryption and decryption of the data. Since the keys might be provided by an IP Author, the encrypting tool, the IP consumer, or possibly even a third party distributor of the IP, a separate specification from **author** and **encrypt\_agent** is necessary. The value of the **data\_keyowner** should be such that either it is a name specific to the decrypting tool, in which case the decrypting tool will use its own embedded key. Otherwise it has to be a value which is available in the tool's key database. So it cannot be an arbitrary value, it has to be from a set of value which are agreed upon by different interacting parties.

A separate specification of the owner of any addition key encryption is provided with the **key\_keyowner** pragma expression below.

ENCRYPTION OUTPUT: The **data\_keyowner** should be unchanged in the output file, except where a digital signature is used in which case it is encrypted with the **key\_method** and found in the **key\_block**.

DECRYPTION INPUT: During decryption, the **data\_keyowner** can be combined with the **data\_keyname** or **data\_public\_key** to determine the appropriate secret/private key to use during decryption of the **data\_block**.

## 28.2.11 data\_method

### 28.2.11.1 Syntax

`data_method=<method_name>`

### 28.2.11.2 Description

ENCRYPTION INPUT: The **data\_method** pragma expression indicates the encryption algorithm that shall be used to encrypt subsequent **begin/end** blocks. The encryption method is an identifier that is commonly associated with a specific encryption algorithm.

This standard specifies the following values for the **data\_method** pragma expression. Additional identifier values are implementation-defined:

DES	Data Encryption Standard
DESX	METHOD_DESX
RSA	RSA Public Key
DSA	Digital Signature Algorithm
RC2	RSA RC2
RC4	RSA RC4
RC5	RSA RC5
RC6	RSA RC6
PGP	Pretty Good Privacy algorithm

Editor' sNote: The above list should be replaced with a normative reference to an existing registry of encryption algorithm identifiers. IETF and W3C are potential registries, and others may exist.

ENCRYPTION OUTPUT: The **data\_method** should be unchanged in the output file, except where a digital signature is used in which case it is encrypted with the **key\_method** and found in the **key\_block**.

DECRYPTION INPUT: The **data\_method** indicates the algorithm that should be used to decrypt the **data\_block**.

### 28.2.12 data\_keyname

#### 28.2.12.1 Syntax

```
data_keyname=<string>
```

#### 28.2.12.2 Description

ENCRYPTION INPUT: The **data\_keyname** pragma expression provides the name of the key or key pair that should be used to decrypt the **data\_block**. A given **data\_keyowner** will typically have multiple keys that they have shared in different ways with different vendors or customers. This pragma expression indicates which of these many keys has been used.

ENCRYPTION OUTPUT: When a **data\_keyname** is provided in the input, it indicates the key that should be used for encrypting the data. The encrypting tool must be able to combine this pragma expression with the **data\_keyowner** and determine the key to use. The **data\_keyname** itself should be output as clear text in the output file except where a digital envelope is used. In case of digital envelope mechanism the **data\_keyname** is encrypted using **key\_method** and **key\_keyname/key\_public\_key** and encoded in the **key\_block**.

DECRYPTION INPUT: In use models where the **data\_keyowner** has provided a secret/private key to a Tool Vendor, or a Tool Vendors secret key has been used, then a unique key name must be identified for each key during this exchange. This key name is then used to identify at decryption time which of many possible secret keys for a given key owner should be used for decryption.

### 28.2.13 data\_public\_key

#### 28.2.13.1 Syntax

data\_public\_key=<key>

#### 28.2.13.2 Description

ENCRYPTION INPUT: The **data\_public\_key** pragma expression indicates that the next line of the file contains the encoded value of the public key, preceded by the single line comment prefix. This is the public key that should be used to encrypt the data. The encoding is specified by the **encoding** pragma expression that is currently in effect. If both **data\_public\_key** and **data\_keyname** are present then they must refer to the same key.

ENCRYPTION OUTPUT: The **data\_public\_key** pragma expression should be output in each protected block for which it is used, followed by the encoded value. The **data\_method** and **data\_public\_key** can be combined to fully specify the required encryption.

DECRYPTION INPUT: The **data\_keyowner** and **data\_method** can be combined with the **data\_public\_key** to determine if the decrypting tool knows the corresponding private key to decrypt a given **data\_block**. If the decrypting tool can compute the required key the model can be decrypted (if licensing allows it).

### 28.2.14 data\_decrypt\_key

#### 28.2.14.1 Syntax

data\_decrypt\_key=<key>

#### 28.2.14.2 Description

ENCRYPTION INPUT: The **data\_decrypt\_key** indicates that the next line contains the encoded value of the key that will decrypt the **data\_block**. This pragma expression should only be used when digital signatures are used. An IP author can generate a key and use it to encrypt the clear text. This encrypted text is then stored in the output file as the **data\_block**. Then the **data\_method** and **data\_decrypt\_key** are encrypted using the **key\_method** and stored in the output file as the contents of the **key\_block**. Note that the **data\_block** itself is not re-encrypted, only the information about the data key is.

ENCRYPTION OUTPUT: The **data\_decrypt\_key** is output as part of the encrypted content of the **key\_block**. The value is encoded as specified by the **encoding** pragma expression.

DECRYPTION INPUT: Upon determining that a digital signature was in use for given protected region, the decrypting tool must decrypt the **key\_block** to find the **data\_decrypt\_key** and **data\_method** which in turn can be used to decrypt the data block.

### 28.2.15 data\_block

#### 28.2.15.1 Syntax

data\_block

#### 28.2.15.2 Description

ENCRYPTION INPUT: A **data\_block** should never be found in an input file unless it is contained within a previously generated **begin\_protected/end\_protected** block in which case it is ignored.

ENCRYPTION OUTPUT: The **data\_block** pragma expression indicates that a data block begins on the next line in the file. Each line of the data block should be preceded by the single line comment prefix. An encrypting tool should take each **begin/end** block, encrypt the contents as specified by the **data\_method** pragma expression, and then encode the block as specified by the **encoding** pragma expression. The resultant text should be output.

DECRYPTION INPUT: The **data\_block** should be first read in the encoded form. The encoding should be reversed, and then the block internally decrypted.

## 28.2.16 digest\_method

### 28.2.16.1 Syntax

```
digest_method=string
```

### 28.2.16.2 Description

ENCRYPTION INPUT: The **digest\_method** pragma expression indicates the MAC algorithm that should be used to generate message digests for subsequent **begin/end** blocks. The string value is an identifier commonly associated with a specific message digest algorithm. The complete set of values will have to be further specified, but may include values such as: MD2, MD5, SHA1.

This standard specifies the following values for the **digest\_method** pragma expression. Additional identifier values are implementation-defined:

MD2	Message Digest Algorithm 2
MD4	Message Digest Algorithm 4
MD5	Message Digest Algorithm 5
SHA1	Secure Hash Algorithm 1

Editor' sNote: The above list should be replaced with a normative reference to an existing registry of message digest algorithm identifiers. IETF and W3C are potential registries, and others may exist.

ENCRYPTION OUTPUT: The **digest\_method** should be unchanged in the output file, except where a digital signature is used in which case it is encrypted with the **key\_method** and found in the **key\_block**.

DECRYPTION INPUT: The **digest\_method** indicates the algorithm that should be used to generate the digest from the **data\_block**.

## 28.2.17 digest\_block

### 28.2.17.1 Syntax

```
digest_block
```

### 28.2.17.2 Description

ENCRYPTION INPUT: If a **digest\_block** pragma expression is found in an input file (other than in a **begin\_protected/end\_protected** block), it should be treated by the encrypting tool as a request to generate a message authentication code in the output file.

ENCRYPTION OUTPUT: A Message Authentication Code (MAC) is used to ensure that the IP has not been modified. In Message Authentication Code, the encrypting tool generates the message digest (fixed length, computationally unique identifier corresponding to a set of data) using the algorithm specified by

the **digest\_method** pragma expression, and encrypts the message digest with the same key used to encrypt the **data\_block**. The encrypted digest is the message authentication code.

This digest should then be encoded using the current **encoding** pragma expression and output on the next line of the output file following the **digest\_block** pragma expression.

DECRYPTION INPUT: In order to authenticate the message, the consuming tool will first decrypt the message, then generate the message digest on the original message, decrypt the message digest with the symmetric key and compare the two message digests. If the two don't match this means that either the MAC or **data\_block** has been altered.

## 28.2.18 key\_keyowner

### 28.2.18.1 Syntax

key\_keyowner=<string>

### 28.2.18.2 Description

ENCRYPTION INPUT: The **key\_keyowner** specifies the company or tool that is providing the keys used for encryption and decryption of the key information. The value of the **key\_keyowner** also has the similar constraint as mentioned in the **data\_keyowner** values.

ENCRYPTION OUTPUT: The **key\_keyowner** should be unchanged in the output file.

DECRYPTION INPUT: During decryption, the **key\_keyowner** can be combined with the **key\_keyname** or **key\_public\_key** to determine the appropriate secret/private key to use during decryption of the **key\_block**.

## 28.2.19 key\_method

### 28.2.19.1 Syntax

key\_method=<method\_name>

### 28.2.19.2 Description

ENCRYPTION INPUT: The **key\_method** pragma expression indicates the encryption algorithm that should be used to encryption the keys used to encrypt the **data\_block**. The same names and formats are used for **data\_method** and **key\_method**. The values have the similar constraint as mentioned for the **data\_method** values.

ENCRYPTION OUTPUT: The **key\_method** should be unchanged in the output file.

DECRYPTION INPUT: The **key\_method** indicates the algorithm that should be used to decrypt the **key\_block**.

## 28.2.20 key\_keyname

### 28.2.20.1 Syntax

key\_keyname=<string>

### 28.2.20.2 Description

ENCRYPTION INPUT: The **key\_keyname** pragma expression provides the name of the key or key pair that should be used to decrypt the **key\_block**. A given **key\_keyowner** will typically have multiple keys that they have shared in different ways with different vendors or customers. This pragma expression indicates which of these many keys has been used.

ENCRYPTION OUTPUT: When a **key\_keyname** is provided in the input, it indicates the key that should be used for encryption the data encryption keys. The encrypting tool must be able to combine this pragma expression with the **key\_keyowner** and determine the key to use. The **key\_keyname** itself should be output as clear text in the output file.

DECRYPTION INPUT: In use models where the **key\_keyowner** has provided a secret/ private key to a Tool Vendor, or a Tool Vendors secret key has been used, then a unique key name must be identified for each key during this exchange. This key name is then used to identify at decryption time which of many possible secret keys for a given key owner should be used for decryption.

### 28.2.21 key\_public\_key

#### 28.2.21.1 Syntax

```
key_public_key=<key_string>
```

#### 28.2.21.2 Description

ENCRYPTION INPUT: The **key\_public\_key** pragma expression indicates that the next line of the file contains the encoded value of the public key, preceded by the single line comment prefix. This is the public key that should be used to encrypt the key data. The encoding is specified by the **encoding** pragma expression that is currently in effect. If both a **key\_public\_key** and **key\_keyname** are present then they must refer to the same key.

ENCRYPTION OUTPUT: The **key\_public\_key** pragma expression should be output in each protected block for which it is used, followed by the encoded value. The **key\_method** and **key\_public\_key** can be combined to fully specify the required encryption of data keys.

DECRYPTION INPUT: The **key\_keyowner** and **key\_method** can be combined with the **key\_public\_key** to determine if the decryption tool knows the corresponding private key to decrypt a given **key\_block**. If the decrypting tool can compute the required key, the data keys can be decrypted.

### 28.2.22 key\_block

#### 28.2.22.1 Syntax

```
key_block
```

#### 28.2.22.2 Description

ENCRYPTION INPUT: A **key\_block** should never be found in an input file unless it is contained within a previously generated **begin\_protected/end\_protected** block in which case it is ignored.

ENCRYPTION OUTPUT: The **key\_block** pragma expression indicates that a key block begins on the next line in the file. Each line of the key block should be preceded by the single line comment prefix. An encrypting tool when requested to use a digital signature, should take any of the **data\_method**, **data\_public\_key**, **data\_keyname**, **data\_decrypt\_key**, and **digest\_block** to form a text buffer. This buffer

should then be encrypted with the appropriate **key\_public\_key** and then the encrypted region should be encoded using the **encoding** pragma expression in effect. The output of this encoding should be generated as the contents of the **key\_block**.

Where more than one **key\_block** pragma expression occurs within a single **begin/end** block, the generated key blocks shall all encode the same data decryption key data. It shall be an error if the data decryption pragma expressions change their value between **key\_block** pragma expressions of a single encryption envelope. Multiple key blocks are specified for the purpose of providing alternative decryption keys for a single decryption envelope.

DECRYPTION INPUT: The **key\_block** should be first read in the encoded form. The encoding should be reversed and then the block internally decrypted. The resulting text can now be parsed to determine the keys required to decrypt the **data\_block**. M

### 28.2.23 decrypt\_license

#### 28.2.23.1 Syntax

```
decrypt_license=<library_name:entry_point_name:string_parameter[:exit_point_name]>
```

#### 28.2.23.2 Description

ENCRYPTION INPUT: The **decrypt\_license** pragma expression will typically be found inside a **begin/end** pair in the original clear text. This is necessary so that it is encrypted in the output IP shipped to the end user.

ENCRYPTION OUTPUT: The **decrypt\_license** is output unchanged in the output description except for encryption and encoding of the pragma exactly as other clear text in the **begin/end** pair. Note that typically it will be output in the **data\_block**.

DECRYPTION INPUT: After encountering a **decrypt\_license** pragma expression in an encrypted model, prior to processing the decrypted text, the application should load the specified library and call the function indicated by the given **entry\_point\_name**, passing it the **string\_parameter** specified. This routine should then return a 0 if the application is licensed to decrypt the model and non-zero if the application is not licensed to decrypt the model. The non-zero value should be printed in any error message about the failure of licensing. If an **exit\_point\_name** is specified then it should be called prior to exiting the decrypting application to allow for releasing the license.

Note that this only provides marginal security because the end-user of the model has the shared library and could use readily available debuggers to debug the calling sequence of the licensing mechanism. They could then produce an equivalent library that returns a 0 but avoids the license check.

### 28.2.24 runtime\_license

#### 28.2.24.1 Syntax

```
runtime_license=<library_name:entry_point_name_name:string_parameter>[:exit_point_name]
```

#### 28.2.24.2 Description

ENCRYPTION INPUT: The **runtime\_license** pragma expression will typically be found inside a **begin/end** pair in the original clear text. This is necessary so that it is encrypted in the output IP shipped to the end user.

ENCRYPTION OUTPUT: The **encrypt\_license** is output unchanged in the output description except for encryption and encoding of the pragma exactly as other clear text in the **begin/end** pair.

DECRYPTION INPUT: After encountering a **runtime\_license** pragma expression in an encrypted model, prior to executing, the application should load the specified library and call the function indicated by the given **entry\_point\_name**, passing it the **string\_parameter** specified. This routine should then return a 0 if the application is licensed to execute the model and non-zero if the application is not licensed to execute the model. The non-zero value should be printed in any error message about the failure of licensing. If an **exit\_point\_name** is specified then it should be called prior to exiting the executing application to allow for releasing the license. Note that execution could mean anything from simulation to layout to synthesis.

Note that this only provides marginal security because the end-user of the model has the shared library and could use readily available debuggers to debug the calling sequence of the licensing mechanism. They could then produce an equivalent library that returns a 0 but avoids the license check.

## 28.2.25 comment

### 28.2.25.1 Syntax

`comment=<value>`

### 28.2.25.2 Description

ENCRYPTION INPUT: The **comment** pragma expression can be found anywhere in an input file and indicates that even if this is found inside a **begin/end** block the value should be output as a comment in clear text in the output immediately prior to the **data\_block**.

This is provided so that comments that may end up being included in other files inside a **begin/end** block can protect themselves from being encrypted. This is important so that critical information such as copyright notices can be explicitly excluded from encryption.

Since this constitutes known clear text that would be found inside the **data\_block** the pragma itself and the value should not be included in the encrypted text.

ENCRYPTION OUTPUT: The entire comment including the beginning pragma should be output in clear text immediately prior to the **data\_block** corresponding to the **begin/end** in which the comment was found.

DECRYPTION INPUT: none

## 28.2.26 reset

### 28.2.26.1 Syntax

`reset`

### 28.2.26.2 Description

ENCRYPTION INPUT: The **reset** pragma expression is used to reset the value of all global pragmas. As only the global pragmas are scope beyond the encryption blocks, after reset all the global pragmas are set to their default values.

Because the scope of pragma definitions is lexical, if an IP author chooses to put common pragmas such as **author** and **author\_info** at the beginning of a list of files, they should include a **reset** pragma at the end of the list of files to ensure that this information is not carried unintentionally into other files.

ENCRYPTION OUTPUT: none

DECRYPTION INPUT: none

## **28.2.27 viewport**

### **28.2.27.1 Syntax**

```
viewport=<object_name>:<access>
```

### **28.2.27.2 Description**

The **viewport** pragma expression describes objects within the current protected envelope for which access should be permitted by the Verilog tool. The specified object name shall be contained within the current envelope. The access value is an implementation specified relaxation of protection

Editor' sNote: The range of access relaxation schemes is not specified in this document, but should be well defined in the final standard.

# Annex A

Editors' Note: This informative annex does not correspond to Annex A in the IEEE 1364 standard. It may be considered as a possible addition for rationale or as a source of informative description for other portions of the final standard.

## A.1. Encryption/Decryption Flow

This section describes the various scenarios which can be used for IP Protection, and it also shows how the relevant pragmas will be used to achieve the desired effect of securely protecting, distributing, and decrypting the model.

The data that needs to be protected from access or from unauthorized modification, should be placed in within the protect **begin/end** block. As the tool encrypts all the information in the **begin/end** block, the information is also protected. Typically the licensing information will be kept to protect against modification.

### A.1.1 Tool Vendor Secret key encryption system

In the secret key encryption system the key is tool vendor proprietary and will be embedded within the tool itself. The same key is used for both encryption and decryption. (In the EDA domain this is the simplest scenario and is roughly equivalent to the existing Verilog-XL ``protect` technique). It has the drawback of being completely tool vendor specific. Using this technique, the IP author can encrypt the IP and any IP consumer with appropriate licenses and the same tool vendor can utilize the IP.

#### A.1.1.1 Encryption Input

The following pragmas are expected when using the tool vendor secret key encryption system. The pragmas required in the encryption input for use of the secret key encryption system are:

**data\_keyname**= <key name> - where <key name> is a valid name of an tool's embedded key.  
**begin/end** - surrounding the region(s) to be encrypted.  
 Additional optional pragmas that may be included are:  
**author**=<string> - to embed author name.  
**author\_info**=<string> - to embed arbitrary author information.  
**data\_keyowner**= <owner identity> - this must be the key owner of the provided name.  
**data\_method**= <method-specifier> - a method appropriate for the given key name.  
 This may be necessary if something other than the default number of rounds, IV, or key width is used.  
**encoding**=<encoding-specifier> - to specify a different encoding.  
**digest\_block** - if a message authorization code is desired to validate that the message has not been modified.  
**decrypt\_license** - if the IP author desires a decryption license.  
**runtime\_license** - if the IP author desires a runtime license.

#### A.1.1.2 Encryption output

The encrypting tool should take the input file and copy all clear text to the corresponding output sections. For each protect begin/end block it should generate:

**begin\_protected** - to start the protected region

**data\_keyowner**= <owner identity>  
**data\_keyname**=<key name>  
**data\_method**=<method-specifier>  
**encoding**=<encoding-specifier>  
**author**=<string> - if provided in the input.  
**author\_info**=<string> - if provided in the input.  
**digest\_block** - followed on the next line(s) by the encoded encrypted digest.  
**data\_block** - followed on the next line(s) by the encoded encrypted data composed of:  
     **decrypt\_license**  
     **encrypt\_license**  
     <text found between begin/end>  
**end\_protected**

## A.1.2 IP Author secret key encryption system

In this mechanism the IP is encrypted with the public key (of public/private key pair) of the IP author, and the decrypting tool will have the IP Author's private key in its secure key database. So only the tool will be able to decrypt the design internally. The IP Authors will have to provide their private keys to the tools' database.

### A.1.2.1 Encryption Input

The following pragmas are expected when using the IP Author secret key encryption system:

**data\_keyname**= <providers key name>  
**begin/end** - surrounding the region(s) to be encrypted.  
 Additional optional pragmas that may be included are:  
**author**=<string> - to embed author name.  
**author\_info**=<string> - to embed arbitrary author information.  
**data\_keyowner**=<owner identity> - this must be the key owner of the provided name.  
**data\_method**= some\_publ\_priv\_encryption\_scheme\_name <method-specifier> - a method appropriate for the given key name.  
 This may be necessary if something other than the default number of rounds, IV, or key width is used.  
**encoding**=<encoding-specifier> - to specify a different encoding.  
**digest\_block** - if a message authorization code is desired to validate that the message has not been modified.  
**decrypt\_license** - if the IP author desires a decryption license.  
**runtime\_license** - if the IP author desires a runtime license.

### A.1.2.2 Encryption output

The encrypting tool should take the input file and copy all clear text to the corresponding output sections. For each protect **begin/end** block it should generate:

**begin\_protected** - to start the protected region  
**data\_keyowner**= <owner identity>  
**data\_keyname**=<providers key name>  
**data\_method**=some\_publ\_priv\_encryption\_scheme\_name  
**encoding**=<encoding-specifier>  
**author**=<string> - if provided in the input.  
**author\_info**=<string> - if provided in the input.  
**digest\_block** - followed on the next line(s) by the encoded encrypted digest.

**viewport.**

**data\_block** - followed on the next line(s) by the encoded encrypted data composed of:

```

decrypt_license
encrypt_license
<text found between begin/end>

```

**end\_protected**

### A.1.3 Digital Envelopes

Editor's Note: This is the preferred exchange form in that it permits use of session keys to limit the amount of cipher text exposure for the exchanged encryption keys. The following text is incorrect in the assumption that asymmetric algorithms are the only useful exchange key mechanisms.

In this mechanism, each user will have a public and private key. The public key is made public while the private key remains secret. The sender encrypts the message using a symmetric key encryption algorithm, then encrypts the symmetric key using the recipient's public key. The recipient then decrypts the symmetric key using the appropriate private key and then decrypts the message with the symmetric key. In this way a fast encryption methods processes large amount of data, yet secret information is never transmitted without encryption. In digital envelopes, using the above encryption technology (secret key encryption system, where the key will be given by the IP author/end user), encryption tool will protect the IP. This symmetric key and algorithm information is them encrypted with a public key, the corresponding private key of which is available to the tool. So only the tool can decrypt the symmetric key internally and decrypt the protected IP.

Instead of using the public key of public/private key pair, a tool specific embedded key can also be used to encrypt the **key\_block**. In this case also as only the tool knows its embedded key, only it can internally decrypt the design, hence the same effect can be achieved. The only disadvantage is that the tool's embedded key will have to be provided to the IP Author in some form.

In the following example the **data\_method** and **data\_keyowner/data\_keyname** are used to encrypt the **data\_block**. The key to encrypt the **data\_block** can either be specified either by a **data\_keyowner/data\_keyname** pair, or by a **data\_decrypt\_key** pragma expression. In the first case, the encrypting tool encrypts the **data\_keyowner** and **data\_keyname** pragmas with the **key\_keymethod/key\_keyname** and puts them in the **key\_block** along with **data\_method**. Alternatively with the **data\_decrypt\_key** pragma, the actual key is provided which is then encrypted with **key\_method/key\_keyname** and stored in the **key\_block**.

In the first approach the **data\_keyowner/data\_keyname** should also be present with the decrypting tool. No such dependency exists with the second approach as the key is present in the file itself.

For better security in the first approach the encrypting tool can actually read the **data\_keyowner/data\_keyname** key and put it in the **key\_block** as **data\_decrypt\_key**. Which not only will remove the dependency mentioned above, but will also protect against the hit & trial breaking of the **data\_block** with the existing keys at the IP users end.

#### A.1.3.1 Encryption Input

The following pragmas are expected when using the Digital Envelopes:

```

key_key_owner = <owner identity>
key_method = some_encryption_scheme_name
key_key_name = <providers key name>

```

**data\_keyname**= <providers key name>  
**begin/end** - surrounding the region(s) to be encrypted.  
 Additional optional pragmas that may be included are:  
**author**=<string> - to embed author name.  
**author\_info**=<string> - to embed arbitrary author information.  
**data\_keyowner**= <owner identity> - this must be the key owner of the provided name.  
**data\_method**= <method-specifier> - a method appropriate for the given key name.  
 This may be necessary if something other than the default number of rounds, IV, or key width is used.  
**encoding**=<encoding-specifier> - to specify a different encoding.  
**digest\_block** - if a message authorization code is desired to validate that the message has not been modified.  
**decrypt\_license** - if the IP author desires a decryption license.  
**runtime\_license** - if the IP author desires a runtime license.

### A.1.3.2 Encryption output

The encrypting tool should take the input file and copy all clear text to the corresponding output sections. For each protect **begin/end** block it should generate:

**begin\_protected** - to start the protected region  
**key\_key\_owner**= <owner identity>  
**key\_method** = some\_encryption\_scheme\_name  
**key\_key\_name**= <providers key name>  
**key\_block** = <encrypted encoded data> this contains the data\_key\_owner,data\_method, and the symmetric data\_key itself in encrypted form  
**encoding**=<encoding-specifier>  
**author**=<string> - if provided in the input.  
**author\_info**=<string> - if provided in the input.  
**digest\_block** - followed on the next line(s) by the encoded encrypted digest.  
**data\_block** - followed on the next line(s) by the encoded encrypted data composed of:  
     **decrypt\_license**  
     **encrypt\_license**  
     <text found between begin/end>  
**end\_protected**