CDN P1364e™/D ~~3~~4
January ~~3~~4, 2005

Draft Standard for Verilog Protected Envelopes

Copyright © 2005 by IEEE.

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

19. Compiler Directives 5

19.10 Pragmas 5

19.10.1 Standard Pragmas 6

28. Protected Envelopes 6

28.1 Processing Protected Envelopes 7

28.1.1 Encryption 7

28.1.2 Decryption 9

28.2 Envelope Directives 9

28.3 Envelope encoding keywords 11

28.3.1 begin 11

28.3.2 end 11

28.3.3 begin_protected 12

28.3.4 end_protected 12

28.3.5 author 13

28.3.6 author_info 13

28.3.7 encrypt_agent 13

28.3.8 encrypt_agent_info 14

28.3.9 encoding 14

28.3.10 data_keyowner 15

28.3.11 data_method 15

28.3.12 data_keyname 17

28.3.13 data_public_key 17

28.3.14 data_decrypt_key 18

28.3.15 data_block 18

Copyright © 2004 IEEE. All rights reserved.

28.3.16 digest_keyowner 19
28.3.17 digest_key_method 19
28.3.18 digest_keyname 20
28.3.19 digest_public_key 20
28.3.20 digest_decrypt_key 20
28.3.21 digest_method 21
28.3.22 digest_block 22
28.3.23 key_keyowner 22
28.3.24 key_method 23
28.3.25 key_keyname 23
28.3.26 key_public_key 23
28.3.27 key_block 24
28.3.28 decrypt_license 24
28.3.29 runtime_license 25
28.3.30 comment 25
28.3.31 reset 26
28.3.32 viewport 26
I.1 Encryption/Decryption Flow 28
I.1.1 Tool Vendor Secret key encryption system 28
I.1.1.1 Encryption Input 28
I.1.1.2 Encryption output 29
I.1.2 IP Author secret key encryption system 29
I.1.2.1 Encryption Input 29
I.1.2.2 Encryption output 29
I.1.3 Digital Envelopes 30
I.1.3.1 Encryption Input 31

Copyright © 2004 IEEE. All rights reserved.

I.1.3.2 Encryption output 31

Copyright © 2004 IEEE. All rights reserved.

Amendment for Verilog Protected Envelopes

19. Compiler Directives

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

19.10 Pragmas

The **`pragma** directive is a structured specification that alters interpretation of the Verilog source. The specification introduced by this directive is referred to as a pragma. The effect of pragmas other than those specified in this standard is implementation-specified.

```
pragma ::=
        `pragma  pragma_name  [ pragma_expression { pragma_expression } ]

pragma_expression ::= pragma_keyword
        | pragma_keyword = pragma_value
        | pragma_value

pragma_value ::= ( pragma_expression { , pragma_expression } )
        | number
        | string
        | identifier

pragma_keyword ::= identifier
```

The pragma specification is identified by the pragma_name which follows the **`pragma** directive. The pragma_name is followed by an optional list of pragma expressions which qualify the altered interpretation indicated by the pragma_name. Unless otherwise specified, pragma directives for pragma_names that are not recognized by an implementation shall have no effect on interpretation of the Verilog source text.

Copyright © 2004 IEEE. All rights reserved.

19.10.1 Standard Pragmas

The **reset** and **resetall** pragmas shall restore the default values and state of pragma_keywords associated with the affected pragmas. These default values shall be those which the tool defines before any Verilog text has been processed. The **reset** pragma shall reset the state for all pragma names which appear as pragma keywords in the directive. The **resetall** pragma shall reset the state of all pragma names recognized by the implementation.

28. Protected Envelopes

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

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. The pragma expressions of the **protect** pragma are evaluated in sequence from left to right. Interpretation of protected envelopes shall not be altered based on whether the sequence of pragma expressions occurs in a single **protect** pragma directive, or in a sequence of **protect** pragma directives. In this clause, unless otherwise specified, pragma directives, pragma keywords, and pragma expressions shall refer to occurrences of **protect** pragma directives and their associated pragma keywords and pragma expressions.

Envelopes may be defined for either of two modes of processing. Encryption envelopes specify the pragma expressions for encrypting source text regions. An encryption envelope begins in the source text when a **begin** pragma expression is encountered. The end of the encryption envelope occurs at the point where an **end** pragma expression is encountered. The end pragma expression is said to close the envelope and shall be associated with the most recent begin pragma that has not already been closed.

Decryption envelopes specify the pragma expressions for decrypting encrypted text regions. A decryption envelope begins in the source text when a **begin_protected** pragma expression is encountered. The end of the decryption envelope occurs at the point where an **end_protected** pragma expression is encountered. The **end_protected** pragma expression is said to close the envelope and shall be associated with the most recent **begin_protected** that has not already been closed. Decryption envelopes may contain other envelopes within their enclosed data block. The number of nested decryption envelopes that can be processed is implementation-specified.

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.

Copyright © 2004 IEEE. All rights reserved.

Content keywords are those pragma expressions which are within the region of text that is processed during encryption or decryption of a protected envelope.

28.1 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 compilation step that follows.

28.1.1 Encryption

Verilog tools that provide encryption services shall transform source text containing encryption envelopes by replacing each encryption envelope 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. The body of the encryption envelope is encrypted using the specified key, referred to as the exchange key, and is recorded in the decryption envelope as a **data_block**.

Tools that provide encryption services may support session keys, to limit exposure to the exchange key which is specified by the IP author using the encryption envelope pragma expressions. A session key is created in a unspecified manner, to encrypt the data from the encryption envelope. A copy of the session key is encrypted using the exchange key and is recorded in a **key_block** 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**.

The following example shows the use of the protect pragma to specify encryption of design data. The encryption method is a simple substitution cipher where each alphabetic character is replaced with the thirteenth character in alphabetic sequence, commonly referred to as “rot13”. Non-alphabetic characters are not substituted. The following design data contains an encryption envelope which specifies the desired protection.

Copyright © 2004 IEEE. All rights reserved.

```
module secret (a, b)
  input a;
  output b;

`pragma protect encoding=(enctype="raw")
`pragma protect runtime_license=(library="libsecret.so", entry="chklic",
match=42 exit="rlslic", match=42)
`pragma protect data_method="x-caesar", data_keyname="rot13", begin
  reg b;

  initial
    begin
      b = 0;
    end

  always
    begin
      #5 b = a;
    end
`pragma protect end

endmodule // secret
```

After encryption processing, the following design data is produced. The decryption envelope is written with a “raw” encoding to make the substitution encryption directly visible.

Copyright © 2004 IEEE. All rights reserved.

```
module secret (a, b)
  input a;
  output b;

`pragma protect begin_protected
`pragma protect data_method="x-caesar", data_keyname="rot13"
`pragma protect data_block encoding=(enctype="raw", bytes=28697)

// 93847520uwzaofxogjj (xabja cynvagrkg frphevgl)

`centzn cebgrpg__ehagvzr_yvprafr=(yvoenel="yvofrperg.fb", ragel="puxyvp",
zngpu=42 rkvg="eyfyvp", zngpu=42)
  ert o;

  vavgvny
    ortva
      o = 0;
    raq

  nyjnlf
    ortva
      #5 o = n;
    raq

// fqxyws902xaf0n09etw (xabja cynvagrkg frphevgl)

`pragma protect end_protected
`pragma reset protected

endmodule // secret
```

Note (informative): Products which include cryptographic algorithms may be subject to government regulations in many jurisdictions. Users of this standard are advised to seek the advice of competent counsel to determine their obligations under those regulations.

28.1.2 Decryption

Verilog tools that support decrypting compilation shall transform source text containing decryption envelopes by replacing each decryption envelope with the decrypted source text 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. Unless otherwise specified, occurrences of the protect pragma directive shall not be included in the decrypted source text which replaces the decryption envelope.

28.2 Envelope Directives

Protected envelopes are lexical regions delimited by protect pragma directives. The effect of a particular protect pragma directive is specified by its pragma expressions. This standard defines the pragma keyword names listed in the following table for use with the protect pragma. These pragma keywords are defined in clause 28.3, with a specification of how each participates in the encryption and decryption processing modes.

Copyright © 2004 IEEE. All rights reserved.

Pragma keyword	*Description*
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	Identifies the author of an envelope
author_info	Specifies additional author information
encrypt_agent	Identifies the encryption service
encrypt_agent_info	Specifies additional encryption service information
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 data encryption session key
data_block	Begins an encoded block of encrypted data
digest_keyowner	Identifies the owner of the digest encryption key
digest_key_method	Identifies the digest encryption algorithm
digest_keyname	Specifies the name of the digest encryption key
digest_public_key	Specifies the public key for digest encryption
digest_decrypt_key	Specifies the digest encryption session key
digest_method	Specifies the digest computation algorithm
digest_block	Specifies a message digest 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 decryption envelope

Copyright © 2004 IEEE. All rights reserved.

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

In protected 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.

28.3 Envelope encoding keywords

28.3.1 begin

28.3.1.1 Syntax

```
begin
```

28.3.1.2 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.

Nesting of pragma **begin**/**end** blocks is erroneous. 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.

ENCRYPTION OUTPUT: The **begin** pragma expression is replaced in the encryption output stream by the **begin_protected** pragma expression. Following **begin_protected** all pragma expressions required as encryption output should be generated prior to the **end_protected** pragma expression. Protected envelopes should be completely self-contained to avoid any undesired interaction when multiple encrypted models exist in the decryption input stream. The **data_block** and **key_block** pragma expressions introduce the encrypted data or keys and will always be found within a **begin_protected**/**end_protected** envelope. All text, including comments and other protect pragmas, occurring between the **begin** pragma expression and the corresponding **end** pragma expression shall, unless otherwise specified, be encrypted and placed in the encryption output stream using the **data_block** pragma expression. An unspecified length of arbitrary comment text may be added by the encrypting tool to the beginning and end of the input text in order to prevent known text attacks on the encrypted content of the **data_block**.

DECRYPTION INPUT: none

28.3.2 end

28.3.2.1 Syntax

```
end
```

Copyright © 2004 IEEE. All rights reserved.

28.3.2 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. The **end** pragma expression is replaced in the encryption output stream by the **end_protected** pragma expression.

ENCRYPTION OUTPUT: none

DECRYPTION INPUT: none

28.3.3 begin_protected

28.3.3.1 Syntax

```
begin_protected
```

28.3.3.2 Description

ENCRYPTION INPUT: When a **begin_protected**/**end_protected** block is found in an input file during encryption, its contents are treated as input clear text. This allows a previously encrypted model to be re-encrypted as a portion of a larger model. Any other protect pragmas inside the **begin_protected**/**end_protected** block shall not be interpreted and shall not override pragmas in effect. Nested encryption must not corrupt pragma values in the current encryption in process.

ENCRYPTION OUTPUT: The **begin_protected** pragma expression, and the entire content of the protected envelope up to the corresponding **end_protect** pragma expression, should be encrypted into the current **data_block** as specified by the current method and keys.

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.

28.3.4 end_protected

28.3.4.1 Syntax

```
end_protected
```

28.3.4.2 Description

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

ENCRYPTION OUTPUT: The **end_protected** pragma expression following the corresponding **begin_protected** pragma expression should be encrypted into the current **data_block** as specified by the current method and keys.

DECRYPTION INPUT: The **end_protected** pragma expression indicates the end of a set of pragmas that should be sufficient to decrypt the current block.

Copyright © 2004 IEEE. All rights reserved.

28.3.5 author

28.3.5.1 Syntax

```
author=<string>
```

28.3.5.2 Description

ENCRYPTION INPUT: The **author** pragma expression specifies a string that identifies the name of the IP author. It is distinct from the comment pragma expression so that this information can be recognized without need for parsing of a comment string value.

ENCRYPTION OUTPUT: If present in the encryption envelope, the **author** pragma expression, shall be placed in a pragma directive enclosed within the protected envelope, but shall not be encrypted into the **data_block**. Otherwise it is copied without change into the output stream.

DECRYPTION INPUT: none.

28.3.6 author_info

28.3.6.1 Syntax

```
author_info=<string>
```

28.3.6.2 Description

ENCRYPTION INPUT: The **author_info** pragma expression specifies a string that contains additional information provided by the IP Author. It is distinct from the comment pragma expression so that this information can be recognized without need for parsing of a comment string value.

ENCRYPTION OUTPUT: If present in the encryption envelope, the **author_info** pragma expression shall be placed in a pragma directive enclosed within the protected envelope, but shall not be encrypted into the **data_block**. Otherwise it is copied without change into the output stream.

DECRYPTION INPUT: none

28.3.7 encrypt_agent

28.3.7.1 Syntax

```
encrypt_agent=<string>
```

28.3.7.2 Description

ENCRYPTION INPUT: none

ENCRYPTION OUTPUT: The **encrypt_agent** pragma expression specifies a string that identifies the name of the encrypting tool. The encrypting tool shall generate this pragma expression and place it in a pragma directive enclosed within the protected envelope, but shall not encrypt it into the **data_block**.

DECRYPTION INPUT: none

Copyright © 2004 IEEE. All rights reserved.

28.3.8 encrypt_agent_info

28.3.8.1 Syntax

```
encrypt_agent_info=<string>
```

28.3.8.2 Description

ENCRYPTION INPUT: none

ENCRYPTION OUTPUT: The **encrypt_agent_info** pragma expression specifies a string that contains additional information provided by the encrypting tool. If provided, the **encrypt_agent_info** pragma expression shall be placed within a pragma directive enclosed within the protected envelope, but shall not be encrypted into the **data_block**.

DECRYPTION INPUT: none

28.3.9 encoding

28.3.9.1 Syntax

```
encoding=(enctype=<string>, line_length=<number>, bytes=<number>)
```

28.3.9.2 Description

ENCRYPTION INPUT: The **encoding** pragma expression specifies how the **data_block**, **digest_block**, and **key_block** content shall be encoded. This encoding ensures that all binary data produced in the encryption process can be treated as text. If an **encoding** pragma expression is present in the input stream it specifies how the output should be encoded.

The **encoding** pragma expression shall be a pragma_expression value containing encoding sub-keywords separated by white space. The following sub-keywords are defined for the value of the **encoding** pragma expression.

enctype=<string> - specifies the method for calculating the encoding. This standard specifies the following identifiers as string values for the enctype sub-keyword. The following identifiers are associated with their respective encoding algorithms. The required methods are standard in every implementation and optional identifiers are implementation-specific but are required to use the following identifiers for the corresponding encoding algorithm. Additional identifier values and their corresponding encoding algorithms are implementation-defined.

enctype		***Encoding Algorithm***
uuencode	REQUIRED	IEEE 1003.1-2001 (uuencode Historical Algorithm)
base64	REQUIRED	IETF RFC 2045 (also IEEE 1003.1 (uuencode -m)
quoted-printable	OPTIONAL	IETF RFC 2045
raw	OPTIONAL	Identity transformation; No encoding shall be performed, and the data may contain non-printable characters.

Copyright © 2004 IEEE. All rights reserved.

line_length=<number> - this is the maximum number of characters (after any encoding) in a single line of the **data_block**. Insertion of line breaks in the **data_block** after encryption and encoding allows the generated text files to be usable by commonly available text tools.

bytes=<number> - this is the number of bytes in the original block of data before any encoding or the addition of line breaks. This encoding keyword shall be ignored in the encryption input.

ENCRYPTION OUTPUT: The **encoding** directive shall 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 tool shall generate an encoding descriptor which specifies in the bytes keyword the number of bytes in the original block of data.

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.

28.3.10 data_keyowner

28.3.10.1 Syntax

```
data_keyowner=<string>
```

28.3.10.2 Description

ENCRYPTION INPUT: The **data_keyowner** specifies the legal entity or tool that provided the keys used for encryption and decryption of the data. This pragma keyword permits use of a third party key, distinct from one associated with either **author** or **encrypt_agent**. The **data_keyowner** value is used by the encrypting tool to select the key used to encrypt the **data_block**. The values for **data_keyname**, **data_decrypt_key**, and **data_public_key** must be unique for the specified **data_keyowner**.

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 placed in a **key_block**.

DECRYPTION INPUT: During decryption, the **data_keyowner** is combined with the **data_keyname** or **data_public_key** to determine the appropriate secret/private key to use during decryption of the **data_block**.

28.3.11 data_method

28.3.11.1 Syntax

```
data_method=<string>
```

Copyright © 2004 IEEE. All rights reserved.

28.3.11.2 Description

ENCRYPTION INPUT: The **data_method** pragma expression specifies 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 identifiers as string values for the data_method pragma expression. The following identifiers are associated with their respective encryption types. The required methods are standard in every implementation and optional identifiers are implementation-specific but are required to use the following identifiers for the corresponding cipher. Additional identifier values and their corresponding ciphers are implementation-defined.

Identifier		***Encryption algorithm***
des-cbc	REQUIRED	DES in CBC mode, see FIPS 46-3, Data Encryption Standard (DES).
3des-cbc	OPTIONAL	three-key 3DES in CBC mode, see FIPS 46-3, Data Encryption Standard (DES); ANSI X9.52-1998 Triple Data Encryption Algorithm Modes of Operation.
aes128-cbc	OPTIONAL	AES with 128-bit key, see FIPS 197, Advanced Encryption Standard (AES).
aes256-cbc	OPTIONAL	AES in CBC mode, with 256-bit key
aes192-cbc	OPTIONAL	AES with 192-bit key
blowfish-cbc	OPTIONAL	Blowfish in CBC mode, see Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish), Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp191-204.
twofish256-cbc	OPTIONAL	Twofish in CBC mode, with 256-bit key, see Schneier, B., et. al. “The Twofish Encryption Algorithm: A 128-bit block cipher, 1st Edition”, Wiley, 1999.
twofish192-cbc	OPTIONAL	Twofish with 192-bit key
twofish128-cbc	OPTIONAL	Twofish with 128-bit key
serpent256-cbc	OPTIONAL	Serpent in CBC mode, with 256-bit key, see Anderson, R., Biham, E., and Knudsen, L. “Serpent : A proposal for the Advanced Encryption Standard”, NIST AES Proposal 1998.; http://www.cl.cam.ac.uk/ftp/users/rja14/serpent.tar.gz
serpent192-cbc	OPTIONAL	Serpent with 192-bit key
serpent128-cbc	OPTIONAL	Serpent with 128-bit key

Copyright © 2004 IEEE. All rights reserved.

Identifier		*Encryption algorithm*
cast128-cbc	OPTIONAL	CAST-128 in CBC mode, see IETF RFC 2144, The CAST-128 Encryption Algorithm.
rsa	OPTIONAL	RSA, see IETF RFC 2313, PKCS #1: RSA Encryption.
Elgamal	OPTIONAL	Elgamal, see Taher ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", IEEE Transactions on Information Theory, v. IT-31, n. 4, 1985, pp469–472
pgp-rsa	OPTIONAL	OpenPGP RSA key, see IETF RFC 2440, OpenPGP Message Format.

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 placed in a **key_block**.

DECRYPTION INPUT: The **data_method** specifies the algorithm that should be used to decrypt the data_block.

28.3.12 data_keyname

28.3.12.1 Syntax

```
data_keyname=<string>
```

28.3.12.2 Description

ENCRYPTION INPUT: The **data_keyname** pragma expression specifies the name of the key, or key pair for an ~~asymetric~~asymmetric encryption algorithm, that should be used to decrypt the **data_block**. It is erroneous to specify a **data_keyname** that is not a member of the list of keys known for the given **data_keyowner.**

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 shall 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: The **data_keyname** value is combined with the **data_keyowner** to select a single key which shall be used to decrypt the **data_block** from the protected envelope.

28.3.13 data_public_key

28.3.13.1 Syntax

```
data_public_key
```

Copyright © 2004 IEEE. All rights reserved.

28.3.13.2 Description

ENCRYPTION INPUT: The **data_public_key** pragma expression specifies that the next line of the file contains the encoded value of the public key to 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).

28.3.14 data_decrypt_key

28.3.14.1 Syntax
`data_decrypt_key`

28.3.14.2 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**.

28.3.15 data_block

28.3.15.1 Syntax
`data_block`

28.3.15.2 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. The encrypting tool should take each **begin**/**end** block, encrypt the

Copyright © 2004 IEEE. All rights reserved.

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.

28.3.16 digest_keyowner

28.3.16.1 Syntax

```
digest_keyowner=<string>
```

28.3.16.2 Description

ENCRYPTION INPUT: The **data_keyowner** specifies the legal entity or tool that provided the keys used for encryption and decryption of the data. This pragma keyword permits use of a third party key, distinct from one associated with either **author** or **encrypt_agent**. The **digest_keyowner** value is used ~~bys~~by the encrypting tool to select the key used to encrypt the **digest_block**. The values for **digest_keyname**, **digest_decrypt_key**, and **digest_public_key** must be unique for the specified **digest_keyowner**. If no **digest_keyowner** is specified in the input, then the default value of **digest_keyowner** shall be the current value of **data_keyowner**.

ENCRYPTION OUTPUT: The **digest_keyowner** should be unchanged in the output file, except where a digital signature is used in which case it is encrypted with the **digest_key_method** and placed in a **digest_key_block**.

DECRYPTION INPUT: During decryption, the **digest_keyowner** is combined with the **digest_keyname** or **digest_public_key** to determine the appropriate secret/private key to use during decryption of the **digest_block**.

28.3.17 digest_key_method

28.3.17.1 Syntax

```
digest_key_method=<string>
```

28.3.17.2 Description

ENCRYPTION INPUT: The **digest_key_method** pragma expression indicates the encryption algorithm that shall be used to encrypt subsequent **digest_block** contents. The values specified for **digest_key_method** to identify encryption algorithms are the same as those specified for **data_method**. If no **digest_key_method** is specified in the input, then the default value of **digest_key_method** shall be the current value of **data_method**.

ENCRYPTION OUTPUT: The **digest_key_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** algorithm and uses the key found in the **key_block**.

DECRYPTION INPUT: The **digest_key_method** indicates the algorithm that should be used to decrypt the **digest_block**.

Copyright © 2004 IEEE. All rights reserved.

28.3.18 digest_keyname

28.3.18.1 Syntax

```
digest_keyname=<string>
```

28.3.18.2 Description

ENCRYPTION INPUT: The **digest_keyname** pragma expression provides the name of the key, or key pair for an ~~asymetric~~asymmetric encryption algorithm, that should be used to decrypt the **digest_block**. It is erroneous to specify a **digest_keyname** that is not a member of the list of keys known for the given **digest_keyowner.** If no **digest_keyname** is specified in the input, then the default value of **digest_keyname** shall be the current value of **data_keyname**.

ENCRYPTION OUTPUT: When a **digest_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 **digest_keyowner** and determine the key to use. The **digest_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 **digest_keyname** is encrypted using **key_method** and **key_keyname**/**key_public_key** and encoded in the **key_block**.

DECRYPTION INPUT: The **digest_keyname** value is combined with the **digest_keyowner** to select a single key which shall be used to decrypt the **digest_block** from the protected envelope**.**

28.3.19 digest_public_key

28.3.19.1 Syntax

```
digest_public_key
```

28.3.19.2 Description

ENCRYPTION INPUT: The **digest_public_key** pragma expression indicates that the next line of the file contains the encoded value of the public key used to encrypt the digest. The encoding is specified by the **encoding** pragma expression that is currently in effect. If both **digest_public_key** and **digest_keyname** are present then they must refer to the same key. If no **digest_public_key** is specified in the input, then the default value of **digest_public_key** shall be the current value of **data_public_key**.

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

DECRYPTION INPUT: The **digest_keyowner** and **digest_key_method** can be combined with the **digest_public_key** to determine if the decrypting tool knows the corresponding private key to decrypt a given **digest_block**. If the decrypting tool can compute the required key the model can be decrypted (if licensing allows it).

28.3.20 digest_decrypt_key

28.3.20.1 Syntax

```
digest_decrypt_key
```

Copyright © 2004 IEEE. All rights reserved.

28.3.20.2 Description

ENCRYPTION INPUT: The **digest_decrypt_key** indicates that the next line contains the encoded value of the key that will decrypt the **digest_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 digest. This encrypted text is then stored in the output file as the **digest_block**. Then the **digest_key_method** and **digest_decrypt_key** are encrypted using the key_method and stored in the output file as the contents of the **key_block**. Note that the **digest_block** itself is not re-encrypted, only the information about the digest key is. If no **digest_decrypt_key** is specified in the input, then the default value of **digest_decrypt_key** shall be the current value of **data_decrypt_key**.

ENCRYPTION OUTPUT: The **digest_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 **digest_decrypt_key** and **digest_key_method** which in turn can be used to decrypt the digest block.

28.3.21 digest_method

28.3.21.1 Syntax

```
digest_method=<string>
```

28.3.21.2 Description

ENCRYPTION INPUT: The **digest_method** pragma expression specifies the message digest algorithm that should be used to generate message digests for subsequent **data_block** and **key_block** output. The string value is an identifier commonly associated with a specific message digest algorithm.

This standard specifies the following values for the **digest_method** pragma expression. Additional identifier values are implementation-defined.

Identifier		***Digest Algorithm***
sha1	REQUIRED	Secure Hash Algorithm, see FIPS 180-1, Secure Hash Standard.
md5	REQUIRED	Message Digest Algorithm 5, see IETF RFC 1321, The MD5 Message-Digest Algorithm.
md2	OPTIONAL	Message Digest Algorithm 2, see IETF RFC 1319, The MD2 Message-Digest Algorithm.
RIPE-MD-160	OPTIONAL	RIPE-MD-160, ISO/IEC 10118-3:2004

Copyright © 2004 IEEE. All rights reserved.

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 placed in a **key_block**.

DECRYPTION INPUT: The **digest_method** indicates the algorithm that should be used to generate the digest from the **data_block**.

28.3.22 digest_block

28.3.22.1 Syntax

```
digest_block
```

28.3.22.2 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 digest in the output file.

ENCRYPTION OUTPUT: A message digest is used to ensure that the encrypted data has not been modified. The encrypting tool generates the message digest (a 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 as specified by the **digest_key_method** pragma keyword, using the key specified by **digest_keyname, digest_key_keyowner**, **digest_public_key**, and **digest_decrypt_key**. If **digest_key_method** is not specified for the encryption envelope, then the current **data_method** encryption key shall be used.

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. A **digest_block** shall be generated for each **key_block** and **data_block** that is generated in the encryption process, and shall immediately follow the **key_block** or **data_block** to which it refers.

DECRYPTION INPUT: In order to authenticate the message, the consuming tool will decrypt the encrypted data, generate a message digest from the decrypted data, decrypt the message digest in the **digest_block** with the specified key and compare the two message digests. If the two digests do not match, then either the **digest_block** or the encrypted data has been altered since the input data was encrypted. The message digest for a **key_block** or **data_block** shall be contained in a **digest_block** immediately following the **key_block** or **data_block**.

28.3.23 key_keyowner

28.3.23.1 Syntax

```
key_keyowner=<string>
```

28.3.23.2 Description

ENCRYPTION INPUT: The **key_keyowner** specifies the legal entity or tool that provided the keys used for encryption and decryption of the key information. The value of the **key_keyowner** also has the same constraints specified for the **data_keyowner** values.

ENCRYPTION OUTPUT: The **key_keyowner** should be unchanged in the output file.

Copyright © 2004 IEEE. All rights reserved.

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**.

28.3.24 key_method

28.3.24.1 Syntax

```
key_method=<string>
```

28.3.24.2 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 values specified for **key_method** to identify encryption algorithms are the same as those specified for **data_method**.

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**.

28.3.25 key_keyname

28.3.25.1 Syntax

```
key_keyname=<string>
```

28.3.25.2 Description

ENCRYPTION INPUT: The **key_keyname** pragma expression provides the name of the key, or key pair for an ~~asymetric~~asymmetric encryption algorithm, that should be used to decrypt the **key_block**. It is erroneous to specify a **key_keyname** that is not a member of the list of keys known for the given **key_keyowner.**

ENCRYPTION OUTPUT: When a **key_keyname** is provided in the input, it indicates the key that should be used for encrypting 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: The **key_keyname** value is combined with the **key_keyowner** to select a single key which shall be used to decrypt the **data_block** from the protected envelope**.**

28.3.26 key_public_key

28.3.26.1 Syntax

```
key_public_key
```

28.3.26.2 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 to 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.

Copyright © 2004 IEEE. All rights reserved.

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.

28.3.27 key_block

28.3.27.1 Syntax

```
key_block
```

28.3.27.2 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. The 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 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** is first read in the encoded form, the encoding is reversed and then the block is internally decrypted. The resulting text is then parsed to determine the keys required to decrypt the **data_block**.

28.3.28 decrypt_license

28.3.28.1 Syntax

```
decrypt_license=(library=<string>, entry=<string>, feature=<string>,
[ exit=<string>, ] [ match=<number> ] )
```

28.3.28.2 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**.

Copyright © 2004 IEEE. All rights reserved.

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 **entry** function, passing it the **feature** specified string. The return value of the **entry** function shall be compared to the the **match** value. If the application is licensed to decrypt the model the returned value shall compare equal to the **match** value,and shall compare non-equal otherwise. If the application is not licensed to decrypt the model no decryption shall be performed, and the application shall produce an error message that includes the return value of the **entry** function. If an **exit** function is specified then it shall be called prior to exiting the decrypting application to allow for releasing the license.

Note: This mechanism only provides limited 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.

28.3.29 runtime_license

28.3.29.1 Syntax

```
runtime_license=(library=<string>, entry=<string>, feature=<string>,
[ exit=<string>,] [ match=<number> ] )
```

28.3.29.2 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 **runtime_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 entry function, passing it the **feature** specified string. The return value of the entry function shall be compared to the match value. If the application is licensed to execute the model, the returned value shall compare equal to the match value, and shall compare non-equal otherwise. If the application is not licensed to execute the model, execution shall not begin and the application shall produce an error message that includes the return value of the entry function. If an **exit** is specified then it shall be called prior to exiting the executing application to allow for releasing the license. Note that execution could mean any evaluation of the model, including simulation, layout, or synthesis.

Note: This mechanism only provides limited 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. IP authors may wish to implement their own licensing scheme embedded within the behavior of the model, possibly using PLI and/or system tasks.

28.3.30 comment

28.3.30.1 Syntax

```
comment=<string>
```

Copyright © 2004 IEEE. All rights reserved.

28.3.30.2 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

28.3.31 reset

28.3.31.1 Syntax

```
reset
```

28.3.31.2 Description

ENCRYPTION INPUT: The **reset** pragma expression is a synonym for a reset pragma directive that contains protect in the pragma keyword list. Following the reset, all protect pragma keywords are restored to their default values.

Because the scope of pragma definitions is lexical and extends from the point of the directive until the end of the compilation input, 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 unintentionally visible in other files.

ENCRYPTION OUTPUT: none

DECRYPTION INPUT: none

28.3.32 viewport

28.3.32.1 Syntax

```
viewport=(object=<string>, access=<string>)
```

28.3.32.2 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.

Copyright © 2004 IEEE. All rights reserved.

Copyright © 2004 IEEE. All rights reserved.

~~Editors's~~ Editor's Note: Add the following new annex following Annex H:

Annex I
(Informative)

I.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.

I.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.

I.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.

Copyright © 2004 IEEE. All rights reserved.

I.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

I.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.

I.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.

I.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

Copyright © 2004 IEEE. All rights reserved.

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

I.1.3 Digital Envelopes

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 then 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 a 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 know 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.

Copyright © 2004 IEEE. All rights reserved.

I.1.3.1 Encryption Input

The following pragmas are expected when using the Digital Envelopes:

key_keyowner = <owner identity>
key_method = some_encryption_scheme_name
key_keyname = <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.

I.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_keyowner = <owner identity>
key_method = some_encryption_scheme_name
key_keyname = <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

Copyright © 2004 IEEE. All rights reserved.