

# An Open IP Encryption Flow Permits Industry-Wide Interoperability



Andrew Dauman, Vice President, Engineering, Synopsys' Synplicity Business Group

## Introduction

Today's extremely large and complex ASIC and FPGA designs use significant amounts of third-party intellectual property (IP). These IP blocks may represent general-purpose processor cores, digital signal processor (DSP) cores, memory controllers, communications functions, etc. Furthermore, this third-party IP, which may account for a large proportion of the overall design, often originates from a number of different IP vendors.

Not surprisingly, due to the fact that each IP block represents a considerable amount of time and investment, the IP vendors wish to guard their secrets. The way this is achieved is to encrypt the source, which means encoding it so as to make it unintelligible to unauthorized parties.

The problem is that, at the time of this writing, there is no standard for encryption and decryption in electronic design flows that facilitates industry-wide interoperability. As a result, different IP vendors and EDA vendors have employed a variety of proprietary schemes. In turn, this has resulted in a huge support burden on the various organizations, it is confusing to the end user, and it can result in a lack of consistency (simulating one version of the IP block and synthesizing a different version, for example).

In order to address this issue, the scientists and engineers at Synopsys' Synplicity Business Group have invented (and implemented) an open IP encryption environment that will facilitate the use of protected IP throughout the design flow: from IP vendor to EDA vendor to silicon vendor.

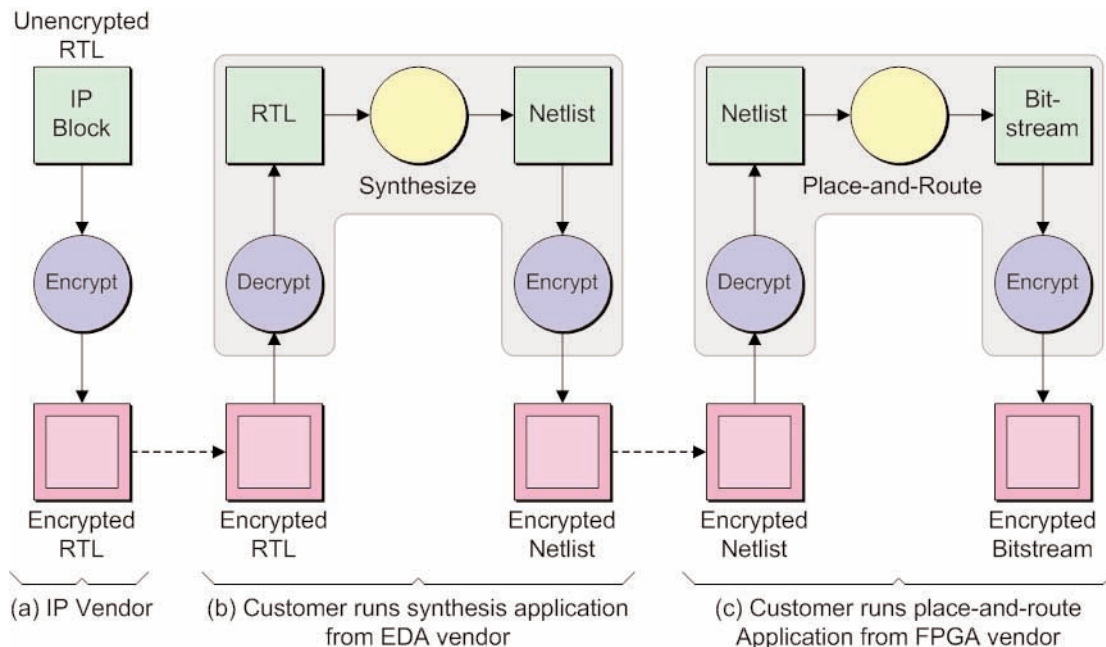
This paper first discusses where the various encryption and decryption steps occur in the design flow. Next, it introduces the conventional encryption techniques – specifically symmetric and asymmetric encryption algorithms – and explains the problems associated with these approaches in the context of an electronic design flow. Finally, a hybrid symmetric-asymmetric open solution is described that leverages existing technology, that fully addresses the needs of modern electronic design environments, and that would be easy to adopt by IP, EDA, and silicon vendors.

## Encryption and Decryption Steps in the Design Flow

Before we introduce the various encryption/decryption options in detail, it's important to understand where these activities occur in the design flow. As an example, let's consider the encryption and decryption activities with regard to the synthesis component of an FPGA flow as illustrated in Figure 1.

The IP block, whose function is typically captured as a Register Transfer Level (RTL) representation, is originally encrypted by the third-party IP vendor (Figure 1a). This encrypted block is subsequently passed to the customer's design team. The design team will typically combine this block with other IP blocks and with its own custom-generated RTL to create a representation of the entire design. The team will then use synthesis tools from an EDA vendor to translate this design into a gate-level netlist (Figure 1b); later, they will use the place-and-route tools provided by the FPGA vendor to generate the configuration bit-stream that will ultimately be used to configure (program) the FPGA (Figure 1c).

Once again, Figure 1 focuses on the synthesis-related activities in the flow. As will be discussed later in this paper, it may be required for the encrypted IP block to be used by multiple EDA tools – such as synthesis and simulation – which are often supplied by different vendors.



**Figure 1. Encryption and decryption activities with respect to the synthesis portion of an FPGA design flow.**

In the case of synthesis, our example block – along with other IP blocks – is first decrypted and then synthesized along with any unprotected portions of the design. In many cases, the ensuing netlist – or at least the portions of the netlist representing the protected IP functions – will be re-encrypted before being fed forward to the FPGA vendor's tools (we will discuss this point in more detail later). It is important to note that all decryption, data manipulation, and encryption activities take place inside the synthesis application itself; the unencrypted IP is never accessible by the user, the encrypted files are decrypted in memory only (no disk storage), and the decryption is performed on small blocks so as to thwart hacking via memory dump techniques.

Following synthesis, the encrypted netlist is passed to the FPGA vendor's tools. Once again, any protected portions of the netlist are first decrypted before being processed by the place-and-route engines. The final step in Figure 1 shows the encryption of the bitstream generated by the silicon vendor's place-and-route application. In this case, the place-and-route tool may encrypt either the entire bitstream or only those portions of the bitstream representing the protected IP. This step falls under the responsibility of the FPGA vendor's tools, it is a function of the underlying FPGA architecture, and it is not further considered in this paper.

An ASIC design flow is very similar to the FPGA flow illustrated in Figure 1. The ASIC flow will, however, include many verification steps that are not present in the FPGA flow; each of these steps will require the use of an EDA tool, and each of these tools may come from a different EDA vendor. Furthermore, in the case of an ASIC flow, the output from the place-and-route engines will be the GDSII files that are used to create the photo masks, which are in turn used to fabricate the device. These files are typically not encrypted at the present time; having said this, the methodology proposed in this paper could easily be applied to this content.

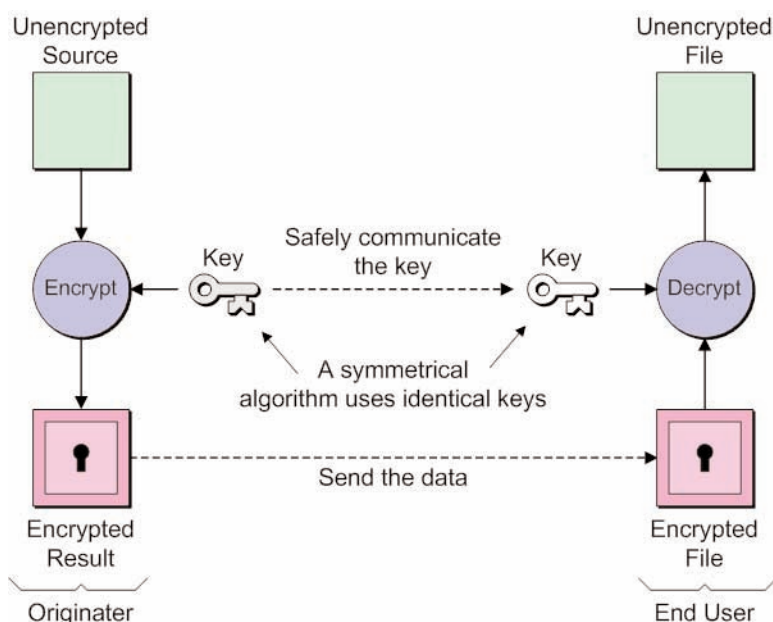
**The Problems with Current Encryption Methodologies**

There are two major classes of encryption/decryption algorithms, which may be classed as symmetric and asymmetric. As will be discussed, each of these techniques has associated advantages and disadvantages.

### Symmetric Encryption

In order to encrypt a source file, the encryption algorithm uses a special number known as the key. The value of this key modifies the detailed operation of the algorithm; that is, the way the contents of the original file will be "scrambled up." This means that if the same file is encrypted using two different keys, the results will be totally dissimilar. Algorithms and keys are created in such a way as to make "cracking the code" by unauthorized parties as difficult as possible. The point is that, in order to open the encrypted file and access its data, the end user also requires access to an appropriate key.

Historically, encryption algorithms have primarily been of a type known as symmetric. This means that the same key is used to encrypt and decrypt the file (Figure 2). The advantage of this technique is speed due to its relatively low computational requirements. Using a modern computer, encrypting even a large file using a symmetric algorithm takes only seconds, and the time taken to decrypt the file when it is accessed by an application such as synthesis is unperceivable to the user.



**Figure 2. A symmetrical encryption algorithm requires the originator to communicate the key to the end user (the folks performing the decryption).**

Examples of this type of algorithm are the Data Encryption Standard (DES), whose specification was first published in 1977, Triple DES [also known as TDES or TDEA (Triple Data Encryption Algorithm), which involves using DES three times], and the more sophisticated Advanced Encryption Standard (AES), which was adopted in 2001.

The Achilles' heel to symmetric encryption is the need to communicate the key. Since the art of cryptography began, this has involved trusted couriers traveling around the world to convey keys to the end users. This approach is obviously not practical in the context of electronic designs. The situation is further confused by the fact that there are so many IP and EDA vendors. Although actual numbers are difficult to tie down, there are probably 300 to 500 significant IP vendors in the world distributing anywhere from 25,000 to 30,000 different IP blocks (of these, approximately 2/3 are digital, while 1/3 are analog and mixed-signal).

If each IP vendor used only a single key, then in the event that key is "leaked" or "cracked," all of the IP from that vendor would be compromised. Thus, in order to maximize security, each IP vendor is obliged to associate a unique key with each EDA vendor. Similarly, each EDA vendor needs to associate a unique key with each

silicon/FPGA vendor. The end result is a morass of confusion, not the least that it's difficult to ensure that the same version of the IP is being used by all of the applications such as simulation and synthesis.

Furthermore, if the IP vendor decides to change a key and reissue that IP, users will have to wait until the next release of the EDA application (which has the keys from the various IP and silicon/FPGA vendors embedded inside it) is in production.

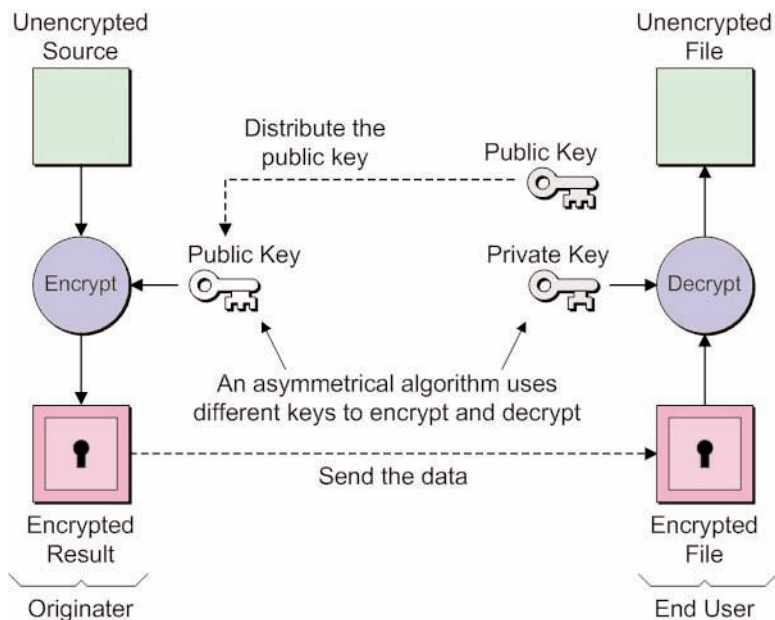
Every time an IP vendor proposes their own proprietary encryption scheme, they have to persuade the various EDA vendors to use it. Similarly, every time an EDA vendor proposes their own proprietary approach, they have to persuade the IP vendors, other EDA vendors, and the silicon/FPGA vendors to adopt it. The end result is that the industry is riddled with a mix of diverse and incompatible schemes, which drastically impede interoperability between the different applications in the design and implementation flow.

### Asymmetric Encryption

In 1976, cryptographer Whitfield Diffie and electrical engineer Martin Hellman created a new form of encryption/decryption known as asymmetric. The "asymmetric" appellation is applied because the key used to decode the data is different to the key used to encode it. Although the DH (Diffie-Hellman) protocol is still used, a more general and more commonly used approach was described by MIT researchers in 1977; this system is known as RSA based on its discoverers' surnames (Rivest, Shamir, and Adleman).

Asymmetric schemes are also commonly known as public key encryption, because they rely on the use of two keys: a public key and a private key. The idea here is that the public key, which is the product of two prime numbers, is made available to everyone (or at least, to everyone who needs to know about it). This public key is used for encryption, but it cannot be used to decrypt the ensuing file; decryption requires access to the private key, which is one of the prime numbers used to create the public key.

For example, an EDA vendor such as Synplicity would create a public key and an associated private key. Synplicity would then make the public key available to all of the IP vendors, who would use it to encrypt their IP blocks. Synplicity would then use its private key (which would be embedded in its applications, and which – for security purposes – would itself be stored in an encrypted form inside these applications) to decrypt the IP from each of the vendors (Figure 3).



**Figure 3. An asymmetrical encryption algorithm requires the end user to generate public and private keys, and to provide the public key to the folks performing the encryption.**

In addition to the fact that they are harder to crack than their symmetric cousins, the main advantage of asymmetric schemes is that the key used to decode the file is not being passed around.

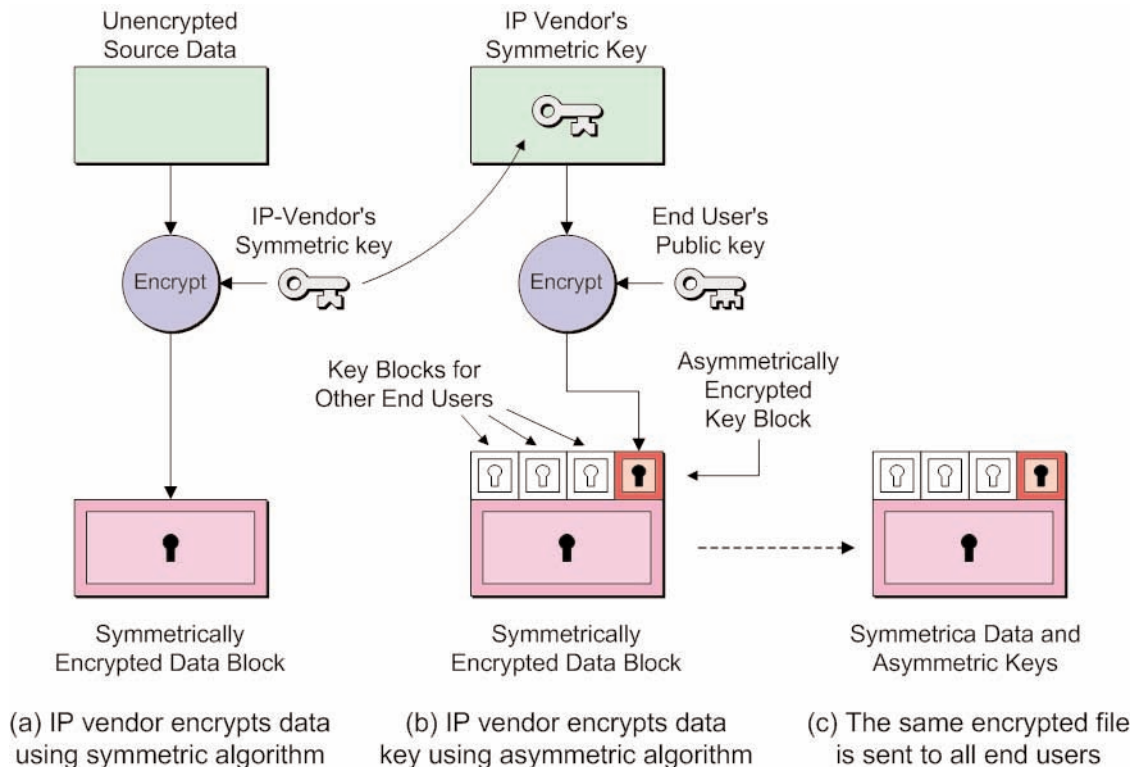
One disadvantage of this scheme is the fact that the IP vendor still has to create multiple encrypted copies of the IP block – one for each EDA vendor – which can result in the consistency problems noted earlier (simulating and synthesizing different versions of the IP, for example).

But perhaps the biggest disadvantage associated with asymmetric approaches is that they are extremely compute-intensive; for example, a large IP block can take several hours to encrypt or decrypt. This is a major inconvenience for the IP provider, and it's essentially a show-stopper for the end user. Consider a design containing several blocks of IP – it would be totally unacceptable to end users for each of these blocks to take several hours to decrypt prior to a simulation or synthesis run being able to commence.

### The Solution: A Hybrid Encryption Flow

The solution to the problem of IP encryption is to employ a hybrid symmetric-asymmetric encryption/decryption flow. This entire solution may be referred to as the cryptosystem. Initially, let's consider the initial encryption performed by the IP vendor (Figure 4).

First, the IP vendor encrypts the IP itself using an internally generated symmetric key (Figure 4a). For the purposes of these discussions, we shall refer to this key as the data key. As was previously discussed, this form of encryption is extremely fast, even on large blocks of IP. The result from this step is known as the data block. It's important to note that the IP vendors are free to select the type of encryption they wish to use: DES, Triple DES, or AES (we shall return to this point a little later).



**Figure 4. In a hybrid approach, the data is encrypted using a symmetric algorithm, and then the key to the encrypted data is itself encrypted using an asymmetrical algorithm.**

Next, the IP vendor takes the data key and encrypts it using the RSA algorithm and the EDA vendor's public key (Figure 4b). The result is known as a key block. The IP vendor repeats this process to generate multiple key

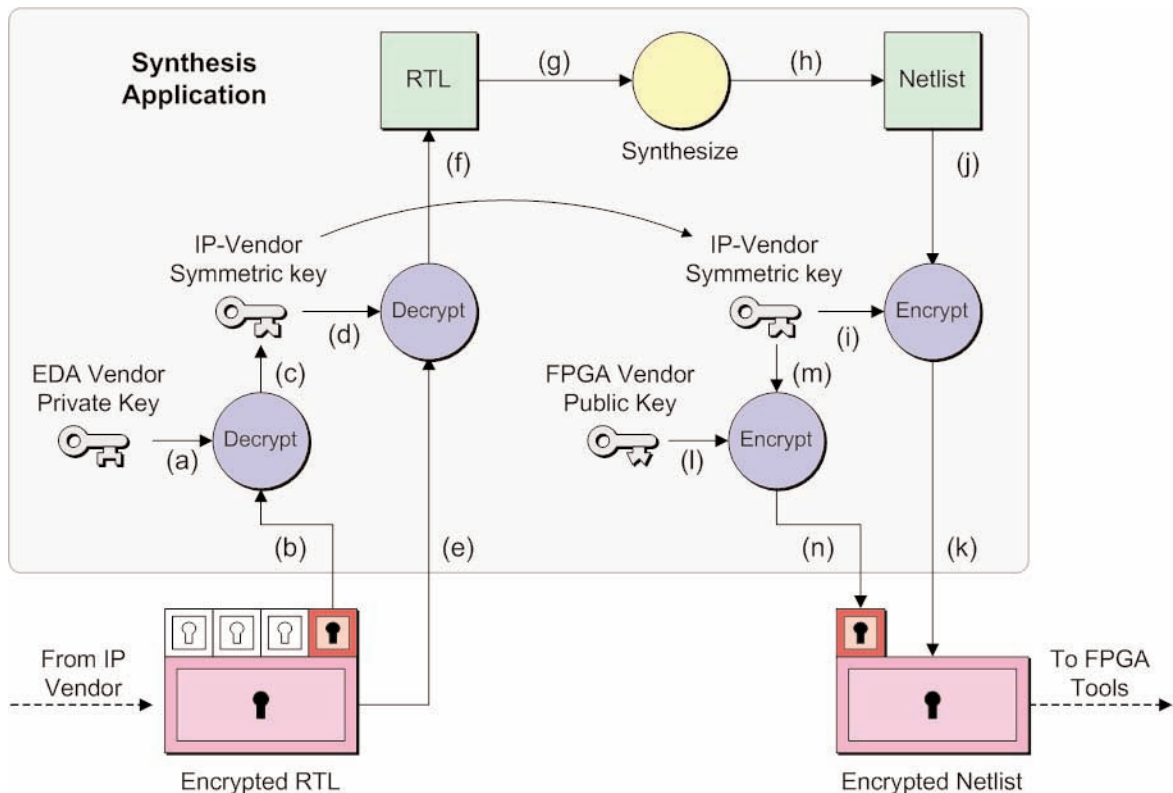
blocks, one for each EDA vendor, where each of these blocks is encrypted using that EDA vendor's public key. Although this form of encryption is inherently compute-intensive, the data key itself is very small, so the entire process takes only a fraction of a second.

Finally, the IP vendor bundles the data block and all of the key blocks into a single file, and communicates this file to all of the EDA vendors (Figure 4c).

Interestingly enough, this hybrid approach is the same technique as that employed by the PGP (Pretty Good Privacy) scheme, which was first published by Phil R. Zimmermann in 1991. However, although PGP is extremely popular, it is not applicable to an electronics design flow because its file format is proprietary and employing this technique would force the use of PGP throughout the flow. Furthermore, PGP is intended only for point-to-point communications and therefore supports only one key block. By comparison, the scheme proposed here supports a multiple-recipient usage model by means of multiple key blocks (note that the recipients are never obliged to share any secrets between each other).

Now let's consider the hybrid flow from the perspective of an EDA vendor. In particular, we shall consider things from the viewpoint of a synthesis vendor, because these actions are a superset of those required from other tools, such as simulation (Figure 5).

When the synthesis tool accesses the IP block from the vendor, the first thing it does is to use the EDA vendor's private key (Figure 5a) – which is encoded and embedded in the synthesis application itself – to decrypt the key block associated with this EDA vendor (Figure 5b) and extract the symmetric data key used by the IP vendor (Figure 5c).



**Figure 5. The synthesis tool decrypts the data key followed by the data; the resulting netlist is encrypted using the original data key, which is itself subsequently encrypted using the FPGA vendor's public key.**

Once extracted, this key (Figure 5d) is used to decrypt the encrypted RTL data block (Figure 5e) so as to access the RTL representation of the IP block (Figure 5f). Note that the synthesis tool will automatically use the appro-

appropriate algorithm to decrypt the data (the algorithm employed by the IP vendor): DES, Triple DES, or AES (once again, we shall return to this point a little later).

Also note that this process will be repeated for any other protected IP blocks from this IP vendor and from other IP vendors. The synthesis tool then takes the RTL from all of the IP blocks along with the unencrypted RTL from the design engineers (Figure 5g) and synthesizes it into a corresponding gate-level netlist (Figure 5h).

Throughout the synthesis process, the synthesis tool keeps track of the protected IP blocks. Assuming the IP vendor has requested that their IP block remains protected, the synthesis tool will use the original symmetric data key supplied by the IP vendor (Figure 5i) – and the original encryption algorithm used by the IP vendor – to take the portion of the netlist associated with this IP block (Figure 5j) and generate a corresponding encrypted netlist data block (Figure 5k).

The synthesis tool next uses a public key supplied by the FPGA vendor (Figure 5l) – which is encoded and embedded in the synthesis application itself – to encrypt the data key (Figure 5m) and generate a corresponding key block (Figure 5n). This process will be repeated for all of the protected IP blocks in the design. Finally, the ensuing netlist is handed over to the FPGA vendor's place and route application. At no time can the contents of any of the protected IP blocks be accessed by unauthorized parties.

### Additional Considerations

This paper is intended only to introduce the high-level concepts underlying Synplicity's proposed hybrid symmetric/asymmetric encryption/decryption technique. More detailed discussions are available in engineering-level documentation. However, there are a few key points that should be noted here for completeness.

In order for EDA design flows to support the proposed cryptosystem, there has to be some mechanism in the appropriate RTL and gate-level netlist files represented in languages such as Verilog, VHDL, and EDIF to describe:

- Multiple key blocks.
- The encryption scheme used for each key block.
- The intended user for each key block.
- The data block.
- The encryption scheme used for the data block.

In fact, Cadence Design Systems has already donated a proposed encryption embedding mechanism to the IEEE 1364-2005 Verilog Working group, which is in the process of defining a standard way (using Verilog pragmas) to embed cryptosystem definitions inside an HDL file. For example, this standard will allow the IP vendor to specify the type of encryption algorithm they wish to employ, such as DES, Triple DES, or AES. (The same underlying approach was also recently donated to the IEEE 1076-200x VHDL technical committee.)

One important aspect of these standards will be some mechanism that allows IP providers to specify attributes that can control how downstream tools are permitted (or required) to use their IP. For example, the IP vendor may require the ability to specify whether or not they wish the portion of the output netlist representing a particular IP block to be re-encrypted. [It is anticipated that some IP blocks may be presented hierarchically in such a way that the core function remains protected throughout the flow, while outer interface functions are permitted to become "visible" (i.e., unencrypted) following synthesis so as to facilitate verification and debugging.]

As part of its efforts, Synplicity has extended the syntax of the proposed schemes to encompass the EDIF format. In the future, this syntax can be extended to embrace other languages such as SystemC. The point is that these pragmas – and this cryptosystem – could potentially be used by any ASCII-based file, such as constraint files.

Also of interest is the fact that the scheme proposed by Synplicity can be employed by anyone in the industry; even very small EDA companies whose resources are severely limited for any activities outside of that company's core mission. Thus, the scheme proposed by Synplicity can employ industry-standard encryption and related

utilities, such as the well-known OpenSSL (Open Secure Socket Layer), which is built into web browsers and operating systems. All that is required is to create a simple Perl script (for example) that takes the information to be encrypted (RTL, or a gate-level netlist, for example) and invokes SSL along with the appropriate encryption algorithm: DES, Triple DES, ASE, or RSA.

### Summary

There is currently no standard for encryption and decryption in electronic design flows that facilitates industry-wide interoperability. As a result, different IP vendors and EDA vendors have employed a variety of proprietary schemes. In turn, this has resulted in a huge support burden on the various organizations, it is confusing to the end user, and it can result in a lack of consistency (simulating one version of the IP block and synthesizing a different version, for example).

In order to address this issue, Synplicity is proposing an open IP encryption environment that will facilitate the use of protected IP throughout the design flow: from IP vendor to EDA vendor to silicon vendor. The advantages of Synplicity's proposed hybrid symmetric/asymmetric encryption/decryption technique are manifold. First, the IP vendor need create only a single version of the encrypted data which is supplied to all interested parties. This ensures consistency, because it guarantees that the same IP will be used by all of the downstream tools.

Another advantage is that fast symmetric encryption can be used for the large data blocks, while slower, more compute-intensive asymmetric algorithms are applied only to the small data keys. But the key advantages to this scheme are that it is open, that it leverages existing technologies, that it fully addresses the needs of modern electronic design environments, that it facilitates industry-wide interoperability, that it is applicable to both ASIC and FPGA design flows, and that it would be easy to adopt by IP, EDA, and silicon/FPGA vendors.

In conclusion, Synplicity proposes that these techniques should be formally endorsed by industry leaders and placed under the auspices of the appropriate standards bodies.

