A Standards-based Approach to IP Protection for HDLs

John Shields
Staff Engineer, Modelsim
Overview

- Introduction
- A Brief Status
- First Look at The Flow
- Encryption Technology Concepts
- Key Management
- Second Look at the Flow
- Examples of the Protect Directives
- A Few Missing Details
- Recommendations for EDA Tool and IP Providers
IP Protection Goals

- Deliver HDL-based IP to potential customers you do not completely trust
- Fast turnaround from design to delivery
- Low cost to protect

Make money while protecting your investment
Some Approaches to IP Protection

- Brick and Mortar Isolation - the Design Center
- Trusting the Customer - Contractual Protection
- Equivalent Models - an abstraction

They all involve compromise in cost, quality, lead time, or effectiveness!

- Encryption – a lossless transformation rendering IP unreadable

  Caution: poor implementation could give false hope
Verilog And VHDL Have Developed A Standard For IP Protection

- Based on well-known standardized methods for strong encryption, encoding, and authentication
- Markup of unprotected HDL source at the token level
- Flexibility to meet issues outside the standards, e.g., international legal considerations
- IEEE Verilog 1364 - 2005
- Accellera VHDL 2006 - pending approval 7/2006
- Minor differences reflect normal syntax considerations and learning curve
Steady-State Flow At IP Vendor

HDL

Markup with protect directives

...protect me...

Encrypt

Deliver

Repeat if business/legal need
The Steady-State Flow At IP Vendor

- Develop your HDL model
- Markup sections of the model with encryption directives
  - optionally sanitize source, obsfuscate it, etc.
- Encrypt the model once
  - with any standards-compliant encryption tool
- Deliver the IP to your customers
  - directly, in a design kit, via a 3rd party, …

Repeat last 3 steps if legal/business considerations dictate different encryption methodology
The Customer Experience

- Customer works in his shop
- IP is only decrypted within the tool and just for its intended purpose
- The tool is obligated not to reveal information that would compromise protected IP
- The encrypted IP is otherwise virtually impossible to crack (OK, we don’t know what the NSA can do :)

Compliant Design Tool
Symmetric Cipher

- A secret key is used to convert plaintext to ciphertext
- The ciphertext can be sent over an insecure channel without concern of compromise
- The same secret key must be used to decrypt ciphertext back into plaintext
- The secret key is delivered in some out of band communication
- Secure bi-directional exchange of messages
- Symmetric ciphers may be implemented in hardware or software and are generally quite fast
Asymmetric Cipher

- A pair of keys - one to encrypt, the other to decrypt
- One key is designated the public key, the other the private key
- Solves key delivery over an insecure channel
- Secure way of delivering messages in one direction
- Very versatile mechanism applied many different ways
- Much slower decryption performance
  …commonly 1000x slower!
Asymmetric Cipher

- To send information only you can read, I'll need your public key
- Guarantee is anything I send to you can only be read by you (…if that's really your public key)
- Anyone else can send you private communication using your public key, too
- You may need to authenticate that this came from me
- Eavesdropping doesn’t hurt much, but “man in the middle” could
Digital Envelopes

- A hybrid approach employing enhanced key security of asymmetric ciphers with the performance of symmetric ciphers
- The data (your IP) is encrypted with a session key using a symmetric cipher
  - One use key, generated in a cryptographically random way
- The session key is placed in a secure digital envelope
  - i.e., encrypted with the public key of an asymmetric cipher
- The ciphertext and the secure envelope are delivered to the user
- The EDA tool decrypts the envelope, gets the session key, and then decrypts the data
- You can send many envelopes holding the same session key at the same time with your data
Digital Signatures

- A signature identifies you
  - Legal significance, weakly resistant to forgery
- In a digital signature, I take a little bit of information about me and/or my message
- To sign, I encrypt it with my *private* key (asymmetric cipher is *not* about which key is revealed)
- With my public key, anyone can authoritatively determine that I signed it
- Two important applications to discuss
  - Managing keys
  - Authenticating delivered IP
Public Key Certificates Build Trust

- A public key and information about its owner that a trusted party digitally signs is called a **public key certificate**.
- The trusted party, the certification authority (CA), guarantees that this is really that owner’s public key.
- The certificate is like a passport and the CA is like your government state department (or notary public).
- The CA can be an external service provider, an industry or government agency.
  - ITU X.509 public key infrastructure standard.
- The signer is trusted by you and supports you in trusting someone else.
- Trust may be built organically using the "**web of trust**" model.
  - (open pgp, key signing parties, key servers..)
Using Signatures for Authentication

- Compute a fixed length binary string from an arbitrarily long message, its *message digest*.
- Cryptographic hash function insures that no 2 messages will have the same digest.
- Decrypt a message, compute its digest, decrypt the signed digest, and compare.
- This process validates the authenticity of the message.
- Using message digests is an option in the standard.
Sanity Check

- Strong encryption is possible with both symmetric and asymmetric ciphers
  - Brute force attacks are computationally infeasible
- Information is secure unless you have the keys
- Key attacks could eavesdrop on key transmission, intercept the key and deliver a fake key, or steal the keys
- Seems it’s all about managing the keys
- Which I claim is about establishing a web of trust and using good quality tools
The Key Management Problem

- The HDL standards do not say how users and tools manage keys
- You, as the IP provider, may have numerous keys
  - 1 per tool vendor, tool, tool rev, user…
  - 1 per model, per model revision, etc.
- You manage these keys inhouse and delivers some of them
- Tool Vendor has the same problems and one more
  - potentially many keys from many IP providers
  - tools must be delivered to customers with secured set of keys available

What does it take to make this manageable?
The Secure Keychain Concept

- In the standard, keys are identified by the tuple of owner and name
- Secure keychain is an encrypted persistent database that accesses a key by its identifiers
- Ability to import, remove, and export keys (maybe some other housekeeping utilities)
- Only the keychain manager enabled tools can access the database
- The concept has a role as a standalone program, part of encryption tool suite, and part of a compliant EDA design tool
A Second Look at the Flow For Producing Protected IP

HDL → Markup with protect directives → "...protect me..." → Encrypt → ...

✓ Reference key(s)

Generate Key(s) → Secure Local Keychain

→ Import → Key
Validate Design Tool

Regression Tests → Compliant Design Tool

Local KeyChain → Export/Import → Tool Specific KeyChain

Runtime Data
Validate EDA Design Tool

- *If necessary*, export keys from your local keychain and import them into a design tool specific local keychain
- Qualify the customer experience with your IP
  - Run validation tests on models using your protected IP with that design tool
- Qualify the tool
  - Inspect all tool outputs for exposure of IP, look at runtime environment for implementation weakness
- Deliver local keychain securely to vendor for hardening into a tool release (its default keychain)
- Option to deliver a tool specific keychain with IP to directly customer?
  - Depends on the strength of protection of the local keychain
Choose Wisely

- The secure keychain is just a concept to discuss principles
- As an IP Provider, to give keys to my vendors I might:
  - generate a key in a file, zip it, and email it…bad idea that often works
  - write it to a memory stick and Fedex it or hand carry it…better
  - Use PKI or web of trust to establish secure communications
- As a tool provider handling my customers keys I might:
  - Allows a tester to copy it to a work directory that anyone can read
  - Write it to a open file that is shipped in “customer_keys” directory
  - Use best practices for handling secured communications and deliver with secure keychain
- The standard has sound mechanisms, but anything can be defeated by careless actions and weak implementation
Simple Secret Key IP Protection Scenario

`protect data_keyowner="ACME IP Provider", data_method="aes192-cbc"

`protect begin

*IP source text...*

`protect end
Simple Secret Key IP Protection Scenario

`protect begin_protected
`protect
  encrypt_agent="Encryptomatic", encrypt_agent_info="2.3.4a"
`protect data_keyowner="ACME IP Provider", data_method="aes192-cbc"
`protect encoding = (encType="base64",
  line_length=40, bytes=4006), data_block
  encoded encrypted IP ...
`protect end_protected
Default IP Protection Scenario

`protect begin
IP source text ...
`protect end

- encryption tool selected key and algorithm
- can only be decrypted by related tools that share key knowledge
- no user generation or exchange of keys, no referencing of them
- Interoperable EDA tool suites would use same defaults
- Hmm…tool might enable the CAD group to configure defaults
Default IP Protection Scenario

```
`protect begin_protected
`protect encrypt_agent="Encryptomatic",
encrypt_agent_info="2.3.4a" `protect
data_keyowner="Electrowizz Tool Co",
data_keyname="crypto-101", data_method="des-
cbc"

`protect encoding = (enctype="base64",
line_length=40, bytes=4006), data_block encoded
encoded encrypted IP ...

`protect end_protected
```
Digital Envelope Protection Scenario

`protect key_keyowner="ACME IP Owner", key_name="For Joe Designer", key_method="rsa", key_block
`protect data_method="aes192-cbc"
`protect begin
IP source text ...
`protect end

- Key_* directives indicate digital envelope
Digital Envelope Protection Scenario

`protect begin_protected
`protect encrypt_agent="Encryptomatic", encrypt_agent_info="2.3.4a"
`protect key_keyowner="ACME IP Owner", key_name=""For Joe Designer",key_method="rsa"
`protect encoding = (enctype="base64", line_length=40, bytes=256),
  key_block
encoded encrypted session key ...
`protect data_method="aes192-cbc"
`protect encoding = (enctype="base64", line_length=40, bytes=4006),
  data_block
encoded encrypted IP ...
`protect end_protected
Multiple Envelopes

\`protect key_keyowner="ACME IP User1", key_method="rsa", key_block
\`protect key_keyowner="ACME IP User2", key_method="elgamal", key_block
\`protect key_keyowner="ACME IP User3", key_method="aes192-cbc", key_block
\`protect data_method="aes192-cbc"
\`protect begin
IP source text ...
\`protect end
A Signed Digital Envelope

`protect key_keyowner="ACME IP User", key_method="rsa", key_block`
`protect data_method="aes192-cbc"`
`protect digest_keyowner="ACME IP Author", digest_key_method="rsa"
`protect digest_method="sha1", digest_block`
`protect begin
IP source text ...
`protect end`
Other Protect Directives in the Standard

- **Viewports**
  - Opens an aspect of your opaque IP for usability
  - Immature aspect of the standard

- **Licensing**
  - Require a license check to access IP
  - Provisioned with marginal security

- **Encoding**
  - Choose method of encoding binary ciphertext for textual representation

- **Documentation**
  - Comments and/or automatic annotation by encryption tools
Standard Details

- DES is the only required cipher method
  - Symmetric algorithm and highly exportable
  - It can be broken by brute force
- SHA-1 and MD5 are only required cryptographic hash functions for computing message digests
  - They are considered to be very good; SHA-1 is the better one
  - Both have been broken, but spoofing IP is not a practical vulnerability
- Provision is made in the syntax for virtually all known and important ciphers, cryptographic hash functions, and encoders
High Quality EDA Tools for IP Protection
Should…

- Implement the standard
- Provide additional market-driven ciphers, cryptographic hash functions, encoders
- Provide utilities for key generation, keychain management, and encryption
- Employ best practices for developing encryption tools
- Deploy tools within the applicable law
- Provide a process for establishing secure communication with IP providers
- Collaborate on developing best practice for IP protection
Recommendations

- Advocate your requirements to the EDA vendor community
- Learn how vendors will safeguard your keys in their house and in the field when the tools are in use
- Qualify tools – tool-specific defaults, interoperability, performance, direct attack, runtime information compromise
- Use good counsel on export laws, use of encryption, etc.
- Make sure compromise doesn’t originate in your house
- …and you’ll make more money with your HDL-based IP!

TRUST ME ;)}
Backup Material

- Tremendous amount of info on the web
  - Cryptographic technology
  - Related export law

- Experiment with Open PGP standard tools
  - FSF’s GnuPG is a good choice
  - No export restrictions on it to my knowledge (except known sponsors of terror)
Symmetric Cipher

- DES is a NIST standard, a 64 bit block cipher, with 56 bit key
- A workhorse with no known structural attacks, brute force methods have cracked in ~22 hours (~30 2.2GHz Opteron years over 5 months was known level of effort at one time)
- It is exportable to non-terrorist countries (but you want a legal opinion, not mine!)
- Interim improvement is triple-DES (which is 3 times slower)
Symmetric Cipher

- AES is the new published NIST standard based on 128 bit blocks and 128, 256, or 512 bit keys
- It was selected from a large field of candidates in a competitive process
- It is unlikely to be cracked in our lifetime
Asymmetric Cipher

- RSA is most commonly used
- Keys of >1024 bits are quite secure, 2048 should give security for decades