A handful of native addons require linking to OpenSSL in one way or another. This introduces a small challenge since node will sometimes bundle OpenSSL statically (the default for node >= v0.8.x), or sometimes dynamically link to the system OpenSSL (default for node <= v0.6.x).

Good native addons should account for both scenarios. It's recommended that you use the `binding.gyp` file provided below as a starting-point for any addon that needs to use OpenSSL:

```
{
  'variables': {
    # node v0.6.x doesn't give us its build variables,
    # but on Unix it was only possible to use the system OpenSSL library,
    # so default the variable to "true", v0.8.x node and up will overwrite it.
    'node_shared_openssl%': 'true'
  },
  'targets': [
    {
      'target_name': 'binding',
      'sources': [
        'src/binding.cc'
      ],
      'conditions': [
        ['node_shared_openssl=="false"', {
          # so when "node_shared_openssl" is "false", then OpenSSL has been
          # bundled into the node executable. So we need to include the same
          # header files that were used when building node.
          'include_dirs': [
            '<(node_root_dir)/deps/openssl/openssl/include'
          ],
          "conditions" : [
            ["target_arch=='ia32'", {
              "include_dirs": [ "<(node_root_dir)/deps/openssl/config/piii" ]
            }],
            ["target_arch=='x64'", {
              "include_dirs": [ "<(node_root_dir)/deps/openssl/config/k8" ]
            }],
            ["target_arch=='arm'", {
              "include_dirs": [ "<(node_root_dir)/deps/openssl/config/arm" ]
            }]
          ]
        }]
      ]
    }
  ]
}
```

This ensures that when OpenSSL is statically linked into `node` then, the bundled OpenSSL headers are included, but when the system OpenSSL is in use, then only those headers will be used.

Windows?

As you can see this baseline `binding.gyp` file only accounts for the Unix scenario. Currently on Windows the situation is a little less ideal. On Windows, OpenSSL is *always* statically compiled into the `node` executable, so ideally it would be possible to use that copy of OpenSSL when building native addons.

Unfortunately it doesn't seem like that is possible at the moment, as there would need to be tweaks made to the generated `node.lib` file to include the openssl glue functions, or a new `openssl.lib` file would need to be created during the node build. I'm not sure which is the easiest/most feasible.

In the meantime, one possible solution is using another copy of OpenSSL, which is what [`node-bcrypt`]() currently does. Adding something like this to your `binding.gyp` file's `"conditions"` block would enable this:

```
    [ 'OS=="win"', {
      'conditions': [
        # "openssl_root" is the directory on Windows of the OpenSSL files.
        # Check the "target_arch" variable to set good default values for
        # both 64-bit and 32-bit builds of the module.
        ['target_arch=="x64"', {
          'variables': {
            'openssl_root%': 'C:/OpenSSL-Win64'
          },
        }, {
          'variables': {
            'openssl_root%': 'C:/OpenSSL-Win32'
          },
        }],
      ],
      'libraries': [
        '-l<(openssl_root)/lib/libeay32.lib',
      ],
      'include_dirs': [
        '<(openssl_root)/include',
      ],
    }]
```

Now you can direct your users to install OpenSSL on Windows from here (be sure to tell them to install the 64-bit version if they're compiling against a 64-bit version of node): http://slproweb.com/products/Win32OpenSSL.html

Also note that both `node-gyp` and `npm` allow you to overwrite that default `openssl_root` variable on the command line:

```
$ node-gyp rebuild --openssl-root="C:\Users\Nathan\Desktop\openssl"
```