Compilation and Installation - OpenSSLWiki

29-37 minutes

The following page is a combination of the `INSTALL` file provided with the OpenSSL library and notes from the field. If you have questions about what you are doing or seeing, then you should consult `INSTALL` since it contains the commands and specifies the behavior by the development team.

OpenSSL uses a custom build system to configure the library. Configuration will allow the library to set up the recursive makefiles from `makefile.org`. Once configured, you use `make` to build the library. You should avoid custom build systems because they often miss details, like each architecture and platform has a unique `opensslconf.h` and `bn.h` generated by `Configure`.

You must use a C compiler to build the OpenSSL library. You cannot use a C++ compiler. Later, once the library is built, it is OK to create user programs with a C++ compiler. But the library proper must be built with a C compiler.

There are two generations of build system. First is the build system used in OpenSSL 1.0.2 and below. The instructions below apply to it. Second is the build system for OpenSSL 1.1.0 and above. The instructions are similar, but not the same. For example, the second generation abandons the monolithic `Configure` and places individual configurations in the `Configurations` directory. Also, the second generation is more platform agnostic and uses templates to produce a final, top level build file (`Makefile`, `descrip.mms`, what have you).

After you configure and build the library, you should always perform a `make test` to ensure the library performs as expected under its self tests. If you are building OpenSSL 1.1.0 and above, then you will also need PERL 5.10 or higher (see `README.PERL` for details).

OpenSSL's build system does not rely upon `autotools` or `libtool`. Also see Why aren't tools like 'autoconf' and 'libtool' used? in the OpenSSL FAQ.

Retrieve source code[edit]

The OpenSSL source code can be downloaded from OpenSSL Source Tarballs or any suitable ftp mirror. There are various versions including stable as well as unstable versions.

The source code is managed via Git. It's referred to as Master. The repository is

> git://git.openssl.org/openssl.git

The source is also available via a GitHub mirror. This repository is updated every 15 minutes.

- Accessing OpenSSL source code via Git

Configuration[edit]

OpenSSL is configured for a particular platform with protocol and behavior options using `Configure` and `config`.

You should avoid custom build systems because they often miss details, like each architecture and platform has a unique `opensslconf.h` and `bn.h` generated by `Configure`.

Supported Platforms[edit]

You can run `Configure LIST` to see a list of available platforms.

```
$ ./Configure LIST
BC-32
BS2000-OSD
BSD-generic32
BSD-generic64
BSD-ia64
BSD-sparc64
BSD-sparcv8
BSD-x86
BSD-x86-elf
BSD-x86_64
Cygwin
Cygwin-x86_64
DJGPP
...
```

If your platform is not listed, then use a similar platform and tune the `$cflags` and `$ldflags` by making a copy of the configure line and giving it its own name. `$cflags` and `$ldflags` correspond to fields 2 and 6 in a configure line. An example of using a similar configure line is presented in Using RPATHs.

Configure & Config[edit]

You use `Configure` and `config` to tune the compile and installation process through options and switches. The difference between is `Configure` properly handles the host-arch-compiler triplet, and `config` does not. `config` attempts to guess the triplet, so it's a lot like autotool's `config.guess`.

You can usually use `config` and it will do the right thing (from Ubuntu 13.04, x64):

```
$ ./config
Operating system: x86_64-whatever-linux2
Configuring for linux-x86_64
Configuring for linux-x86_64
    no-ec_nistp_64_gcc_128 [default]  OPENSSL_NO_EC_NISTP_64_GCC_128 (skip dir)
    no-gmp          [default]  OPENSSL_NO_GMP (skip dir)
    no-jpake        [experimental] OPENSSL_NO_JPAKE (skip dir)
    no-krb5         [krb5-flavor not specified] OPENSSL_NO_KRB5
    ...
```

Mac OS X can have issues (it's often a neglected platform), and you will have to use `Configure`:

```
 ./Configure darwin64-x86_64-cc
Configuring for darwin64-x86_64-cc
    no-ec_nistp_64_gcc_128 [default]  OPENSSL_NO_EC_NISTP_64_GCC_128 (skip dir)
    no-gmp          [default]  OPENSSL_NO_GMP (skip dir)
    no-jpake        [experimental] OPENSSL_NO_JPAKE (skip dir)
    no-krb5         [krb5-flavor not specified] OPENSSL_NO_KRB5
    ...
```

You can also configure on Darwin by exporting `KERNEL_BITS`:

```
$ export KERNEL_BITS=64
$ ./config shared no-ssl2 no-ssl3 enable-ec_nistp_64_gcc_128 --openssldir=/usr/local/ssl/macosx-
x64/
Operating system: i686-apple-darwinDarwin Kernel Version 12.5.0: Sun Sep 29 13:33:47 PDT 2013;
root:xnu-2050.48.12~1/RELEASE_X86_64
Configuring for darwin64-x86_64-cc
Configuring for darwin64-x86_64-cc
    no-gmp          [default]  OPENSSL_NO_GMP (skip dir)
    no-jpake        [experimental] OPENSSL_NO_JPAKE (skip dir)
    no-krb5         [krb5-flavor not specified] OPENSSL_NO_KRB5
    ...
```

If you provide a option not known to configure or ask for help, then you get a brief help message:

```
$ ./Configure --help
Usage: Configure [no-<cipher> ...] [enable-<cipher> ...] [experimental-<cipher> ...]
[-Dxxx] [-lxxx] [-Lxxx] [-fxxx] [-Kxxx] [no-hw-xxx|no-hw] [[no-]threads] [[no-]shared]
[[no-]zlib|zlib-dynamic] [no-asm] [no-dso] [no-krb5] [sctp] [386] [--prefix=DIR]
[--openssldir=OPENSSLDIR] [--with-xxx[=vvv]] [--test-sanity] os/compiler[:flags]
```

And if you supply an unknown triplet:

```
$ ./Configure darwin64-x86_64-clang
Configuring for darwin64-x86_64-clang
Usage: Configure [no-<cipher> ...] [enable-<cipher> ...] [experimental-<cipher> ...]
[-Dxxx] [-lxxx] [-Lxxx] [-fxxx] [-Kxxx] [no-hw-xxx|no-hw] [[no-]threads] [[no-]shared]
[[no-]zlib|zlib-dynamic] [no-asm] [no-dso] [no-krb5] [sctp] [386] [--prefix=DIR]
[--openssldir=OPENSSLDIR] [--with-xxx[=vvv]] [--test-sanity] os/compiler[:flags]
```

```
pick os/compiler from:
BC-32 BS2000-OSD BSD-generic32 BSD-generic64 BSD-ia64 BSD-sparc64 BSD-sparcv8
BSD-x86 BSD-x86-elf BSD-x86_64 Cygwin Cygwin-pre1.3 DJGPP MPE/iX-gcc OS2-EMX
...

NOTE: If in doubt, on Unix-ish systems use './config'.
```

Dependencies[edit]

If you are prompted to run `make depend`, then you must do so. For OpenSSL 1.0.2 and below, it's required to update the standard distribution once configuration options change.

```
Since you've disabled or enabled at least one algorithm, you need to do
the following before building:

        make depend
```

OpenSSL 1.1.0 and above performs the dependency step for you, so you should not see the message. However, you should perform a `make clean` to ensure the list of objects files is accurate after a reconfiguration.

Configure Options[edit]

OpenSSL has been around a long time, and it carries around a lot of cruft. For example, from above, SSLv2 is enabled by default. SSLv2 is completely broken, and you should disable it during configuration. You can disable protocols and provide other options through `Configure` and `config`, and the following lists some of them.

Note: if you specify a non-existent option, then the configure scripts will proceed without warning. For example, if you inadvertently specify **no-sslv2** rather than **no-ssl2 no-ssl3**, the script will configure *with* SSLv2 and *without* warning for the unknown no-sslv2.

Note: when building a shared object, both the static archive and shared objects are built. You do not need to do anything special to build both when **shared** is specified.

OpenSSL Library Options

Option	Description
--prefix=XXX	See PREFIX and OPENSSLDIR in the next section (below).
--openssldir=XXX	See PREFIX and OPENSSLDIR in the next section (below).
-d	Debug build of the library. Optimizations are disabled (no `-O3` or similar) and `libefence` is used (`apt-get install electric-fence` or `yum install electric-fence`). TODO: Any other features?
shared	Build a shared object in addition to the static archive. You probably need a RPATH when enabling `shared` to ensure `openssl` uses the correct `libssl` and `libcrypto` after installation.
enable-ec_nistp_64_gcc_128	Use on little endian platforms when GCC supports `__uint128_t`. ECDH is about 2 to 4 times faster. Not enabled by default because `Configure` can't determine it. Enable it if your compiler defines `__SIZEOF_INT128__`, the CPU is little endian and it tolerates unaligned data access.
enable-capieng	Enables the Microsoft CAPI engine on Windows platforms. Used to access the Windows Certificate Store. Also see Using Windows certificate store through OpenSSL on the OpenSSL developer list.
no-ssl2	Disables SSLv2. `OPENSSL_NO_SSL2` will be defined in the OpenSSL headers.
no-ssl3	Disables SSLv3. `OPENSSL_NO_SSL3` will be defined in the OpenSSL headers.
no-comp	Disables compression independent of `zlib`. `OPENSSL_NO_COMP` will be defined in the OpenSSL headers.
no-idea	Disables IDEA algorithm. Unlike RC5 and MDC2, IDEA is enabled by default
no-asm	Disables assembly language routines (and uses C routines)
no-dtls	Disables DTLS in OpenSSL 1.1.0 and above
no-dtls1	Disables DTLS in OpenSSL 1.0.2 and below

Option	Description
no-shared	Disables shared objects (only a static library is created)
no-hw	Disables hardware support (useful on mobile devices)
no-engine	Disables hardware support (useful on mobile devices)
no-threads	Disables threading support.
no-dso	Disables the OpenSSL DSO API (the library offers a shared object abstraction layer). If you disable DSO, then you must disable Engines also
no-err	Removes all error function names and error reason text to reduce footprint
no-npn/no-nextprotoneg	Disables Next Protocol Negotiation (NPN). Use `no-nextprotoneg` for 1.1.0 and above; and `no-npn` otherwise
no-psk	Disables Preshared Key (PSK). PSK provides mutual authentication independent of trusted authorities, but it's rarely offered or used
no-srp	Disables Secure Remote Password (SRP). SRP provides mutual authentication independent of trusted authorities, but it's rarely offered or used
no-ec2m	Used when configuring FIPS Capable Library with a FIPS Object Module that only includes prime curves. That is, use this switch if you use `openssl-fips-ecp-2.0.5`.
no-weak-ssl-ciphers	Disables RC4. Available in OpenSSL 1.1.0 and above.
-DXXX	Defines XXX. For example, `-DOPENSSL_NO_HEARTBEATS`.
-DPEDANTIC	Defines PEDANTIC. The library will avoid some undefined behavior, like casting an unaligned byte array to a different pointer type. This define should be used if building OpenSSL with undefined behavior sanitizer (`-fsanitize=undefined`).
-DOPENSSL_USE_IPV6=0	Disables IPv6. Useful if OpenSSL encounters incorrect or inconsistent platform headers and mistakenly enables IPv6. Must be passed to `Configure` manually.
-DNO_FORK	Defines NO_FORK. Disables calls to `fork`. Useful for operating systems like AppleTVOS, WatchOS, AppleTVSimulator and WatchSimulator.
-L*something*, -l*something*, -K*something*, -Wl,*something*	Linker options, will become part of LDFLAGS.
-*anythingelse*, +*anythingelse*	Compiler options, will become part of CFLAGS.

Note: on older OSes, like CentOS 5, BSD 5, and Windows XP or Vista, you will need to configure with `no-async` when building OpenSSL 1.1.0 and above. The configuration system does not detect lack of the Posix feature on the platforms.

Note: you can verify compiler support for `__uint128_t` with the following:

```
# gcc -dM -E - </dev/null | grep __SIZEOF_INT128__
#define __SIZEOF_INT128__ 16
```

PREFIX and OPENSSLDIR[edit]

`--prefix` and `--openssldir` control the configuration of installed components. The behavior and interactions of `--prefix` and `--openssldir` are slightly different between OpenSSL 1.0.2 and below, and OpenSSL 1.1.0 and above.

The ***rule of thumb*** to use when you want something that "just works" for all recent versions of OpenSSL, including OpenSSL 1.0.2 and 1.1.0, is:

- specify *both* `--prefix` and `--openssldir`
- set `--prefix` and `--openssldir` to the same location

One word of caution is *avoid* `--prefix=/usr` when OpenSSL versions are ***not*** binary compatible. You will replace the distro's version of OpenSSL with your version of OpenSSL. It will most likely break everything, including the package management system.

OpenSSL 1.0.2 and below

It is usually *not* necessary to specify `--prefix`. If `--prefix` is *not* specified, then `--openssldir` is used. However, specifying *only* `--prefix` may result in broken builds because the 1.0.2 build system attempts to build in a FIPS configuration.

You can *omit* If `--prefix` and use `--openssldir`. In this case, the paths for `--openssldir` will be used during configuration. If `--openssldir` is not specified, the the default `/usr/local/ssl` is used.

The takeaway is `/usr/local/ssl` is used by default, and it can be overridden with `--openssldir`. The rule of thumb applies for path overrides: specify *both* `--prefix` and `--openssldir`.

OpenSSL 1.1.0 and above

OpenSSL 1.1.0 changed the behavior of install rules. You should specify both `--prefix` and `--openssldir` to ensure `make install` works as expected.

The takeaway is `/usr/local/ssl` is used by default, and it can be overridden with *both* `--prefix` and `--openssldir`. The rule of thumb applies for path overrides: specify *both* `--prefix` and `--openssldir`.

Debug Configuration[edit]

From the list above, it's possible to quickly configure a "debug" build with `./config -d`. However, you can often get into a more amicable state *without* the Electric Fence dependency by issuing:

```
$ ./config no-asm -g3 -O0 -fno-omit-frame-pointer -fno-inline-functions
Operating system: x86_64-whatever-linux2
Configuring for linux-x86_64
Configuring OpenSSL version 1.1.0-pre5-dev (0x0x10100005L)
    no-asm          [option]   OPENSSL_NO_ASM
    ...
Configuring for linux-x86_64
IsMK1MF       =no
CC            =gcc
CFLAG         =-Wall -O3 -pthread -m64 -DL_ENDIAN  -g3 -O0 -fno-omit-frame-pointer
...
```

Don't be alarmed about both `-O3` and `-O0`. The last setting *"sticks"*, and that's the `-O0`.

If you are working in Visual Studio and you can't step into library calls, then see Step into not working, but can force stepping after some asm steps on Stack Overflow.

Modifying Build Settings[edit]

Sometimes you need to work around OpenSSL's selections for building the library. For example, you might want to use `-Os` for a mobile device (rather than `-O3`), or you might want to use the `clang` compiler (rather than `gcc`).

In case like these, its' often easier to modify `Configure` and `Makefile.org` rather than trying to add targets to the configure scripts. Below is a patch that modifies `Configure` and `Makefile.org` for use under the iOS 7.0 SDK (which lacks `gcc` in `/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/`):

- Modifies `Configure` to use `clang`
- Modifies `Makefile.org` to use `clang`
- Modifies `CFLAG` to use `-Os`
- Modifies `MAKEDEPPROG` to use `$(CC) -M`

Setting and resetting of `LANG` is required on Mac OSX to work around a `sed` bug or limitation.

```
OLD_LANG=$LANG
unset LANG

sed -i "" 's|\"iphoneos-cross\"\,\"llvm-gcc\:-O3|\"iphoneos-cross\"\,\"clang\:-Os|g' Configure
sed -i "" 's/CC= cc/CC= clang/g' Makefile.org
sed -i "" 's/CFLAG= -O/CFLAG= -Os/g' Makefile.org
sed -i "" 's/MAKEDEPPROG=makedepend/MAKEDEPPROG=$(CC) -M/g' Makefile.org

export LANG=$OLD_LANG
```

After modification, be sure to dclean and configure again so the new settings are picked up:

```
make dclean

./config
make depend
make all
...
```

Using RPATHs[edit]

RPATH's are supported by default on the BSD platforms, but not others. If you are working on Linux and compatibles, then you have to manually add an RPATH. One of the easiest ways to add a RPATH is to configure with it as shown below.

```
./config -Wl,-rpath=/usr/local/ssl/lib
```

For modern Linux you should also use `-Wl,--enable-new-dtags`. The linker option sets a RUNPATH as opposed to a RPATH. A RUNPATH allows library path overrides at runtime, while a RPATH does not.

```
./config -Wl,-rpath=/usr/local/ssl/lib -Wl,--enable-new-dtags
```

Note well: you should use a RPATH or RUNPATH when building both OpenSSL and your program. If you don't add a RPATH or RUNPATH to both, then your program could runtime-link to the wrong version of OpenSSL. Linking against random versions of a security library is *not* a good idea.

You can also add an RPATH or RUNPATH by hard coding the RPATH into a configure line. For example, on Debian x86_64 open the file `Configure` in an editor, copy `linux-x86_64`, name it `linux-x86_64-rpath`, and make the following change to add the `-rpath` option. Notice the addition of `-Wl,-rpath=...` in two places.

```
"linux-x86_64-rpath", "gcc:-m64 -DL_ENDIAN -O3 -Wall -Wl,-rpath=/usr/local/ssl/lib::
  -D_REENTRANT::-Wl,-rpath=/usr/local/ssl/lib -ldl:SIXTY_FOUR_BIT_LONG RC4_CHUNK DES_INT
DES_UNROLL:
  ${x86_64_asm}:elf:dlfcn:linux-shared:-fPIC:-m64:.so.\$(SHLIB_MAJOR).\$(SHLIB_MINOR):::64",
```

Above, fields 2 and 6 were changed. They correspond to `$cflag` and `$ldflag` in OpenSSL's builds system.

Then, Configure with the new configuration:

```
$ ./Configure linux-x86_64-rpath shared no-ssl2 no-ssl3 no-comp \
    --openssldir=/usr/local/ssl enable-ec_nistp_64_gcc_128
```

Finally, after `make`, verify the settings stuck:

```
$ readelf -d ./libssl.so | grep -i -E 'rpath|runpath'
0x000000000000000f (RPATH)              Library rpath: [/usr/local/ssl/lib]
$ readelf -d ./libcrypto.so | grep -i rpath
0x000000000000000f (RPATH)              Library rpath: [/usr/local/ssl/lib]
$ readelf -d ./apps/openssl | grep -i rpath
0x000000000000000f (RPATH)              Library rpath: [/usr/local/ssl/lib]
```

Once you perform `make install`, then `ldd` will produce expected results:

```
$ ldd /usr/local/ssl/lib/libssl.so
linux-vdso.so.1 =>  (0x00007ffceff6c000)
 ibcrypto.so.1.0.0 => /usr/local/ssl/lib/libcrypto.so.1.0.0 (0x00007ff5eff96000)
...

$ ldd /usr/local/ssl/bin/openssl
linux-vdso.so.1 =>  (0x00007ffc30d3a000)
libssl.so.1.0.0 => /usr/local/ssl/lib/libssl.so.1.0.0 (0x00007f9e8372e000)
libcrypto.so.1.0.0 => /usr/local/ssl/lib/libcrypto.so.1.0.0 (0x00007f9e832c0000)
...
```

FIPS Capable Library[edit]

If you want to use FIPS validated cryptography, you download, build and install the FIPS Object Module (`openssl-fips-2.0.5.tar.gz`) according to the FIPS User Guide 2.0 and FIPS 140-2 Security Policy. You then download, build and install the FIPS Capable Library (`openssl-1.0.1e.tar.gz`).

When configuring the FIPS Capable Library, you must use `fips` as an option:

```
./config fips <other options ...>
```

If you are configuring the FIPS Capable Library with only prime curves (`openssl-fips-ecp-2.0.5.tar.gz`), then you must configure with `no-ec2m`:

```
./config fips no-ec2m <other options ...>
```

Compile Time Checking[edit]

If you disable an option during configure, you can check if it's available through `OPENSSL_NO_*` defines. OpenSSL writes the configure options to `<openssl/opensslconf.h>`. For example, if you want to know if SSLv3 is available, then you would perform the following in your code:

```
#include <openssl/opensslconf.h>
...

#if !defined(OPENSSL_NO_SSL3)
  /* SSLv3 is available */
#endif
```

Compilation[edit]

After configuring the library, you should run `make`. If prompted, there's usually no need to `make depend` since you are building from a clean download.

Quick[edit]

```
./config <options ...> --openssldir=/usr/local/ssl
make
make test
sudo make install
```

Various options can be found examining the `Configure` file (there is a well commented block at its top). OpenSSL ships with SSLv2, SSLv3 and Compression enabled by default (see `my $disabled`), so you might want to use `no-ssl2 no-ssl3`, `no-ssl3`, and `no-comp`.

Platfom specific[edit]

Linux[edit]

Intel[edit]

ARM[edit]

X32 (ILP32)[edit]

X32 uses the 32-bit data model (ILP32) on x86_64/amd64. To properly configure for X32 under current OpenSSL distributions, you must use `Configure` and use the x32 triplet:

```
# ./Configure LIST | grep x32
linux-x32
...
```

Then:

```
# ./Configure linux-x32
Configuring OpenSSL version 1.1.0-pre6-dev (0x0x10100006L)
    no-asan         [default]  OPENSSL_NO_ASAN (skip dir)
    ...
Configuring for linux-x32
CC            =gcc
CFLAG         =-Wall -O3 -pthread -mx32 -DL_ENDIAN
SHARED_CFLAG  =-fPIC
...
```

If using an amd64-compatible processor and GCC with that supports `__uint128_t`, then you usually add `enable-ec_nistp_64_gcc_128` in addition to your other flags.

Windows[edit]

3noch wrote a VERY good guide in 2012 here (PLEASE NOTE: the guide was written in 2012 and is no longer available at the original location; the link now points to an archived version at the Internet Archive Wayback Machine).

Like he said in his article, make absolutely sure to create separate directories for 32 and 64 bit versions.

W32 / Windows NT - Windows 9x[edit]

type INSTALL.W32

- you need Perl for Win32. Unless you will build on Cygwin, you will need ActiveState Perl, available from http://www.activestate.com/ActivePerl.
- one of the following C compilers:
  - Visual C++
  - Borland C
  - GNU C (Cygwin or MinGW)
- Netwide Assembler, a.k.a. NASM, available from http://nasm.sourceforge.net/ is required if you intend to utilize assembler modules. Note that NASM is now the only supported assembler.

W64[edit]

Read first the INSTALL.W64 documentation note containing some specific 64bits information. See also INSTALL.W32 that still provides additonnal build information common to both the 64 and 32 bit versions.

You may be surprised: the 64bit artefacts are indeed output in the out32* sub-directories and bear names ending *32.dll. Fact is the 64 bit compile target is so far an incremental change over the legacy 32bit windows target. Numerous compile flags are still labelled "32" although those do apply to both 32 and 64bit targets.

The important pre-requisites are to have PERL available (for essential file processing so as to prepare sources and scripts for the target OS) and of course a C compiler like Microsoft Visual Studio for C/C++. Also note the procedure changed at OpenSSL 1.1.0 and is more streamlined. Also see Why there is no ms\do_ms.bat after perl Configure VC-WIN64A on Stack Overflow.

OpenSSL 1.1.0[edit]

For OpenSSL 1.1.0 and above perform these steps:

1. Ensure you have perl installed on your machine (e.g. ActiveState or Strawberry), and available on your %PATH%
2. Ensure you have NASM installed on your machine, and available on your %PATH%
3. Extract the source files to your folder, here `cd c:\myPath\openssl`
4. Launch Visual Studio tool x64 Cross Tools Command prompt
5. Goto your defined folder `cd c:\myPath\openssl`
6. Configure for the target OS with `perl Configure VC-WIN64A` or other configurations to be found in the INSTALL file (e.g. UNIX targets).
    1. For instance: `perl Configure VC-WIN64A`.
7. (*Optional*) In case you compiled before on 32 or 64-bits, make sure you run `nmake clean` to prevent trouble across 32 and 64-bits which share output folder.
8. Now build with: `nmake`
9. Output can be found in the **root** of your folder as **libcrypto-1_1x64.dll** and **libssl-1_1-x64.dll** (with all the build additionals such as .pdb .lik or static .lib). You may check this is true 64bit code using the Visual Studio tool 'dumpbin'. For instance `dumpbin /headers libcrypto-1_1x64.dll | more`, and look at the FILE HEADER section.
10. Test the code using the 'test' make target, by running `nmake test`.
11. Reminder, clean your code to prevent issues the next time you compile for a different target. See step 7.

Windows CE[edit]

Not specified

OpenSSL 1.0.2[edit]

For OpenSSL 1.0.2 and earlier the procedure is as follows.

1. Ensure you have perl installed on your machine (e.g. ActiveState or Strawberry), and available on your %PATH%
2. Ensure you have NASM installed on your machine, and available on your %PATH%
3. launch a Visual Studio tool x64 Cross Tools Command prompt
4. change to the directory where you have copied openssl sources `cd c:\myPath\openssl`
5. configure for the target OS with the command `perl Configure VC-WIN64A`. You may also be interested to set more configuration options as documented in the general INSTALL note (for UNIX targets). For instance: `perl Configure VC-WIN64A`.
6. prepare the target environment with the command: `ms\do_win64a`
7. ensure you start afresh and notably without linkable products from a previous 32bit compile (as 32 and 64 bits compiling still share common directories) with the command: `nmake -f ms\ntdll.mak clean` for the DLL target and `nmake -f ms\nt.mak clean` for static libraries.
8. build the code with: `nmake -f ms\ntdll.mak` (respectively `nmake -f ms\nt.mak`)
9. the artefacts will be found in sub directories out32dll and out32dll.dbg (respectively out32 and out32.dbg for static libraries). The libcrypto and ssl libraries are still named libeay32.lib and ssleay32.lib, and associated includes in inc32 ! You may check this is true 64bit code using the Visual Studio tool 'dumpbin'. For instance `dumpbin /headers out32dll/libeay32.lib | more`, and look at the FILE HEADER section.
10. test the code using the various *test.exe programs in out32dll. Use the 'test' make target to run all tests as in `nmake -f ms\ntdll.mak test`
11. we recommend that you move/copy needed includes and libraries from the "32" directories under a new explicit directory tree for 64bit applications from where you will import and link your target applications, similar to that explained in INSTALL.W32.

Windows CE[edit]

OS X[edit]

The earlier discussion presented a lot of information (and some of it had OS X information). Here are the TLDR versions to configure, build and install the library.

If configuring for 64-bit OS X, then use a command similar to:

```
./Configure darwin64-x86_64-cc shared enable-ec_nistp_64_gcc_128 no-ssl2 no-ssl3 no-comp --
openssldir=/usr/local/ssl/macos-x86_64
make depend
sudo make install
```

If configuring for 32-bit OS X, then use a command similar to:

```
./Configure darwin-i386-cc shared no-ssl2 no-ssl3 no-comp --openssldir=/usr/local/ssl/macosx-i386
make depend
sudo make install
```

If you want to build a multiarch OpenSSL library, then see this answer on Stack Overflow: Build Multiarch OpenSSL on OS X.

iOS[edit]

The following builds OpenSSL for iOS using the iPhoneOS SDK. The configuration avoids the dynamic library the DSO interface and engines.

If you run `make install`, then the headers will be installed in `/usr/local/openssl-ios/include` and libraries will be installed in `/usr/local/openssl-ios/lib`.

32-bit[edit]

For OpenSSL 1.1.0 and above, a 32-bit iOS cross-compiles uses the `ios-cross` target, and options similar to `--prefix=/usr/local/openssl-ios`.

```
$ export CC=clang;
$ export
CROSS_TOP=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer
$ export CROSS_SDK=iPhoneOS.sdk
$ export
PATH="/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin:$PATH"
```

```
$ ./Configure ios-cross no-shared no-dso no-hw no-engine --prefix=/usr/local/openssl-ios

Configuring OpenSSL version 1.1.1-dev (0x10101000L)
    no-afalgeng     [forced]   OPENSSL_NO_AFALGENG
    no-asan         [default]  OPENSSL_NO_ASAN
    no-dso          [option]
    no-dynamic-engine [forced]
    ...
    no-weak-ssl-ciphers [default]  OPENSSL_NO_WEAK_SSL_CIPHERS
    no-zlib         [default]
    no-zlib-dynamic [default]
Configuring for ios-cross

PERL          =perl
PERLVERSION   =5.16.2 for darwin-thread-multi-2level
HASHBANGPERL  =/usr/bin/env perl
CC            =clang
CFLAG         =-O3 -D_REENTRANT -arch armv7 -mios-version-min=6.0.0 -isysroot
$(CROSS_TOP)/SDKs/$(CROSS_SDK) -fno-common
CXX           =c++
CXXFLAG       =-O3 -D_REENTRANT -arch armv7 -mios-version-min=6.0.0 -isysroot
$(CROSS_TOP)/SDKs/$(CROSS_SDK) -fno-common
DEFINES       =NDEBUG OPENSSL_THREADS OPENSSL_NO_DYNAMIC_ENGINE OPENSSL_PIC OPENSSL_BN_ASM_MONT
OPENSSL_BN_ASM_GF2m SHA1_ASM SHA256_ASM SHA512_ASM AES_ASM BSAES_ASM GHASH_ASM ECP_NISTZ256_ASM
POLY1305_ASM
...
```

If you are working with OpenSSL 1.0.2 or below, then use the `iphoneos-cross` target.

```
$ export CC=clang;
$ export
CROSS_TOP=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer
$ export CROSS_SDK=iPhoneOS.sdk
$ export
PATH="/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin:$PATH"

$ ./Configure iphoneos-cross no-shared no-dso no-hw no-engine --prefix=/usr/local/openssl-ios
Configuring for iphoneos-cross
    no-dso          [option]
    no-engine       [option]   OPENSSL_NO_ENGINE (skip dir)
    no-gmp          [default]  OPENSSL_NO_GMP (skip dir)
    no-hw           [option]   OPENSSL_NO_HW
    ...
    no-weak-ssl-ciphers [default]  OPENSSL_NO_WEAK_SSL_CIPHERS (skip dir)
    no-zlib         [default]
    no-zlib-dynamic [default]
IsMK1MF=0
CC            =clang
CFLAG         =-DOPENSSL_THREADS -D_REENTRANT -O3 -isysroot $(CROSS_TOP)/SDKs/$(CROSS_SDK) -
fomit-frame-pointer -fno-common
...
```

64-bit[edit]

For OpenSSL 1.1.0 , a 64-bit iOS cross-compiles uses the `ios64-cross` target, and `--prefix=/usr/local/openssl-ios64`. `ios64-cross`. There is no built-in 64-bit iOS support for OpenSSL 1.0.2 or below.

```
$ export CC=clang;
$ export
CROSS_TOP=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer
$ export CROSS_SDK=iPhoneOS.sdk
$ export
```

```
PATH="/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin:$PATH"

$ ./Configure ios64-cross no-shared no-dso no-hw no-engine --prefix=/usr/local/openssl-ios64

Configuring OpenSSL version 1.1.1-dev (0x10101000L)
    no-afalgeng     [forced]   OPENSSL_NO_AFALGENG
    no-asan         [default]  OPENSSL_NO_ASAN
    no-dso          [option]
    no-dynamic-engine [forced]
    ...
    no-weak-ssl-ciphers [default]  OPENSSL_NO_WEAK_SSL_CIPHERS
    no-zlib         [default]
    no-zlib-dynamic [default]
Configuring for ios64-cross

PERL          =perl
PERLVERSION   =5.16.2 for darwin-thread-multi-2level
HASHBANGPERL  =/usr/bin/env perl
CC            =clang
CFLAG         =-O3 -D_REENTRANT -arch arm64 -mios-version-min=7.0.0 -isysroot
$(CROSS_TOP)/SDKs/$(CROSS_SDK) -fno-common
CXX           =c++
CXXFLAG       =-O3 -D_REENTRANT -arch arm64 -mios-version-min=7.0.0 -isysroot
$(CROSS_TOP)/SDKs/$(CROSS_SDK) -fno-common
DEFINES       =NDEBUG OPENSSL_THREADS OPENSSL_NO_DYNAMIC_ENGINE OPENSSL_PIC OPENSSL_BN_ASM_MONT
SHA1_ASM SHA256_ASM SHA512_ASM VPAES_ASM ECP_NISTZ256_ASM POLY1305_ASM
...
```

Android[edit]

Visit Android and FIPS Library and Android.

More[edit]

VAX/VMS[edit]

I you wonder what are files ending with .com like test/testca.com those are VAX/VMX scripts. This code is still maintained.

OS/2[edit]

NetWare[edit]

5.x 6.x

HP-UX[edit]

HP-UX Itanium FIPS and OpenSSL build

Autoconf[edit]

OpenSSL uses its own configuration system, and does not use Autoconf. However, a number of popular projects use both OpenSSL and Autoconf, and it would be useful to detect either `OPENSSL_init_ssl` or `SSL_library_init` from `libssl`. To craft a feature test for OpenSSL that recognizes both `OPENSSL_init_ssl` and `SSL_library_init`, you can use the following.

```
if test "$with_openssl" = yes ; then
  dnl Order matters!
  if test "$PORTNAME" != "win32"; then
     AC_CHECK_LIB(crypto, CRYPTO_new_ex_data, [], [AC_MSG_ERROR([library 'crypto' is required for
OpenSSL])])
     FOUND_SSL_LIB="no"
     AC_CHECK_LIB(ssl, OPENSSL_init_ssl, [FOUND_SSL_LIB="yes"])
     AC_CHECK_LIB(ssl, SSL_library_init, [FOUND_SSL_LIB="yes"])
     AS_IF([test "x$FOUND_SSL_LIB" = xno], [AC_MSG_ERROR([library 'ssl' is required for
OpenSSL])])
  else
```

```
     AC_SEARCH_LIBS(CRYPTO_new_ex_data, eay32 crypto, [], [AC_MSG_ERROR([library 'eay32' or
'crypto' is required for OpenSSL])])
     FOUND_SSL_LIB="no"
     AC_SEARCH_LIBS(OPENSSL_init_ssl, ssleay32 ssl, [FOUND_SSL_LIB="yes"])
     AC_SEARCH_LIBS(SSL_library_init, ssleay32 ssl, [FOUND_SSL_LIB="yes"])
     AS_IF([test "x$FOUND_SSL_LIB" = xno], [AC_MSG_ERROR([library 'ssleay32' or 'ssl' is required
for OpenSSL])])
  fi
fi
```

Many thanks to the Postgres folks for donating part of their `configure.in`. Also see How to tell Autoconf “require symbol A or B” from LIB? on Stack Overflow.