Botan Build Guide

Jack Lloyd (lloyd@randombit.net)

November 19, 2003

Contents

1 Introduction

This document describes how to build Botan on Unix and MS Windows systems. The Unix oriented descriptions should apply to most common Unix systems today, including Unix-like systems such as BeOS and Plan 9 (and MacOS X). Currently, systems other than Windows and Unix (for example, VMS, MacOS 9, and OS/390) are not supported by the build system, primarily due to lack of access. Please contact the maintainer if you would like to build Botan on such a system.

2 Building the Library

The first step is to run `configure.pl`, which is a Perl script that creates various directories, config files, and a Makefile for building everything. It is run as `./configure.pl CC-OS-CPU <extra args>`. The script requires at least Perl 5.005, and preferably 5.6 or higher.

The tuple CC-OS-CPU specifies what system Botan is being built for, in terms of the C++ compiler, the operating system, and the CPU model. For example, to use GNU C++ on a FreeBSD box that has an Alpha EV6 CPU, one would use “gcc-freebsd-alphaev6”, and for Visual C++ on Windows with a Pentium II, “msvc-windows-pentium2”. To get the list of values for `CC`, `OS`, and `CPU` that `configure.pl` supports, run it with the “`--help`” option.

You can put basically anything reasonable for CPU: the script knows about a large number of different architectures, their sub-models, and common aliases for them. The script does not display all the possibilities in it’s help message because there are simply too many entries (if you’re curious about what exactly is available, you can look at the `%ARCH`, `%ARCH_ALIAS`, and `%SUBMODEL_ALIAS` hashes at the start of the script). You should only select the 64-bit version of a CPU (like “sparc64” or “mips64”) if your operating system knows how to handle 64-bit object code – a 32-bit kernel on a 64-bit CPU will generally not like 64-bit code. For example, gcc-solaris-sparc64 will not work unless you’re running a 64-bit Solaris kernel (for 32-bit Solaris running on an UltraSPARC system, you want gcc-solaris-sparc32-v9). You may or may not have to install 64-bit versions of libc and related system libraries as well.

The script also knows about the various extension modules available. You can enable one or more with the option “`--modules=MOD`”, where `MOD` is some name that identifies the extension (or a comma separated list of them). Modules provide additional capabilities which require the use of non portable APIs. You should enable any extensions which makes sense for your system.

Not all OSes or CPUs have specific support in `configure.pl`. If the CPU architecture of your system isn’t supported by `configure.pl`, use ’generic’. This setting disables machine-specific optimization flags. Similarly, setting OS to ’generic’ disables things which depend greatly on OS support (specifically, shared libraries).

However, it’s impossible to guess which options to give to a system compiler. Thus, if you want to compile Botan with a compiler which `configure.pl` does not support, the script will have to be updated. Preferably, mail the man pages (or similar documentation) for the C and C++ compilers and the system linker to the author, or download the Botan-config package from the Botan web site, and do it yourself. Modifying `configure.pl` on it’s own is useless aside from one-off hacks, because the script is auto-generated by *another* Perl script, which reads a little mini-language that tells it all about the systems in question.

The script tries to guess what kind of makefile to generate, and it almost always guesses correctly (basically, Visual C++ uses NMAKE with Windows commands, and everything else uses Unix make with Unix commands). Just in case, you can override it with `--make-style=somestyle`. The styles Botan currently knows about are ’unix’ (normal Unix makefiles), and ’nmake’, the make variant commonly used by Windows compilers.

2.1 Unix / BeOS / Plan9

The basic build procedure on Unix and Unix-like systems is:

```
$ ./configure.pl CC-OS-CPU --module-set=[unix|beos] --modules=<other mods>
$ make
$ make check # optional, but a good idea
$ make install
```

The 'unix' module set should work on most Unix systems out there (including MacOS X), while the 'beos' module is specific to BeOS. While the two sets share a number of modules, some normal Unix ones don't work on BeOS (in particular, BeOS doesn't have a working **mmap** function), and BeOS has a few extras just for it. Other important modules you might want to include are `comp_zlib` (which wraps `zlib` for use in the library), and, if your hardware supports it, `mp_asm64`. That module is only usable on 64-bit systems, and provides a 20-300% increase in the performance of public key algorithms when used. There is another module, `mp_gmp`, which uses GNU MP for certain low-level operations, and also provides very substantial speedups. You can use both at once without any problems.

The `make install` target has a default directory in which it will install Botan (on everything that's a real Unix, it's `/usr/local`). You can override this by using the `--prefix` argument to `configure.pl`, like so:

```
./configure.pl --prefix=/opt <other arguments>
```

On Unix, the makefile has to decide who should own the files once they are installed. By default, it uses `root:root`, but on some systems (for example, MacOS X), there is no `root` group. Also, if you don't have root access on the system you will want them to be installed owned by something other than root (like yourself). You can override the defaults at install time by setting the `OWNER` and `GROUP` variables from the command line.

```
make OWNER=lloyd GROUP=users install
```

2.2 MS Windows

The situation is not much different here. We'll assume you're using Visual C++ (for Cygwin, the Unix instructions are probably more relevant). You need to have a copy of Perl installed, and have both Perl and Visual C++ in your path.

```
> perl configure.pl msvc-windows-<CPU> [--module-set=win32]
> nmake
> nmake check # optional, but recommended
```

The additional argument `--module-set=win32` is optional, but highly recommended. This will include a set of modules for doing useful things on Windows systems, particularly gathering entropy for the PRNG. In the future, support for high resolution timers, mutexes, and maybe even secure memory allocation will also be made available by this flag.

For Win95 pre OSR2, the `es_capi` module will not work, because CryptoAPI didn't exist. All versions of NT4 lack the ToolHelp32 interface, which is how `es_win32` does it's slow polls, so a version of the library built with that module will not load under NT4. Later systems (98/ME/2000/XP) support both methods, so this shouldn't be much of an issue.

If you have zlib or libbz2 installed, you can also use the `comp_zlib` or `comp_bzip2` modules. The `mp_gmp` module should also work, if you have GNU MP installed. Most of the other modules rely on libraries or system calls not available on Windows, or an inline assembler format not supported by Visual C++.

Unfortunately, there currently isn't an install script usable on Windows. Basically all you have to do is copy the newly created `libbotan.lib` to someplace where you can find it later (say, `C:\Botan\`). Then copy the entire `include\botan` directory, which was constructed when you built the library, into the same directory.

When building your applications, all you have to do is tell the compiler to look for both include files and library files in `C:\Botan`, and it will find both.

2.3 Configuration Parameters

There are some configuration parameters which you may want to tweak before building the library. These can be found in `config.h`. This file is overwritten every time the configure script is run (and does not exist until after you run the script for the first time).

Also included in `config.h` are macros which are defined if one or more extensions are available. All of them begin with `BOTAN_EXT_`. For example, if `BOTAN_EXT_COMPRESSOR_BZIP2` is defined, then an application using Botan can include `<botan/bzip2.h>` and use the Bzip2 filters.

`MP_WORD_BITS`: This macro controls the size of the words used for calculations with the MPI implementation in Botan. You can choose 8, 16, 32, or 64, with 32 being the default. You can use 8, 16, or 32 bit words on any CPU, but the value should be set to the same size as the CPU's registers for best performance. You can only use 64-bit words if the `mp_asm64` module is used; this offers vastly improved performance of public key algorithms on certain 64-bit CPUs (it will set this to 64 automatically when used).

Note that `MP_WORD_BITS` is a macro that *doesn't* start with the typical `BOTAN_` prefix (mostly for historical reasons).

`BOTAN_VECTOR_OVER_ALLOCATE`: The memory container `SecureVector` will over-allocate requests by this amount (in elements). In several areas of the library, we grow a vector fairly often. By over-allocating by a small amount, we don't have to do allocations as often (which is good, because the allocators can be quite slow). If you *really* want to reduce memory usage, set it to 0. Otherwise, the default should be perfectly fine.

`BOTAN_DEFAULT_BUFFER_SIZE`: This constant is used as the size of buffers throughout Botan. A good rule of thumb would be to use the page size of your machine. The default should be fine for most, if not all, purposes.

3 Building Applications

3.1 Unix

Botan usually links in several different system libraries (such as `librt` and `libz`), depending on which modules are configured at compile time. In many environments, particularly ones using static libraries, an application has to link against the same libraries as Botan for the linking step to succeed. But how does it figure out what libraries it *is* linked against?

The answer is to ask the `botan-config` script. This basically solves the same problem all the other `*-config` scripts solve, and in basically the same manner.

There are 4 options:

`--prefix[=DIR]`: If no argument, print the prefix where Botan is installed (such as `/opt` or `/usr/local`). If an argument is specified, other options given with the same command will execute as if Botan as actually installed at `DIR` and not where it really is; or at least where `botan-config` thinks it really is. I should mention that it

`--version`: Print the Botan version number.

`--cflags`: Print options that should be passed to the compiler whenever a C++ file is compiled.

`--libs`: Print options that should be passed to the compiler when the application is linked.

Your `Makefile` can run `botan-config` and get the options necessary for getting your application to compile and link, regardless of whatever crazy libraries Botan might be linked against.

3.2 MS Windows

No special help exists for building applications on Windows. However, given that typically Windows software is distributed as binaries, this is less of a problem - only the developer needs to worry about it. As long as they can remember where they installed Botan, they just have to set the appropriate flags in their Makefile/project file.