BAE SYSTEMS

SMP Setup Manual

Version 2.5

29 March 2005

BAE_IT_AJ_SMP_00013

BAE Systems Information Technology
141 National Business Parkway, Suite 210
Annapolis Junction, Maryland 20701
(301) 953-3600

Revision History

Version	Date	Author	Scope
1.0	9 February 2001		Initial version
2.0	31 October 2001		SNACC/SFL 2.0
2.0.1	1 May 2002		Added ACL Info
2.1	14 June 2002		Updated for release and added details for delivery build shells
2.5	29 March 2005	Sue Beauchamp, Tom Horvath	Modified for new release

Approval History

Version	Date	Approved by
2.5	29 March 2005	Susan Joseph

Table of Contents

1. Introduction

This manual describes the installation procedures for the freeware security libraries developed by BAE Systems Information Technology as part of the Secure Messaging Protocols (SMP) project. The freeware security libraries addressed are Enhanced SNACC ASN.1 Compiler/Library, S/MIME Freeware Library (SFL), Certificate Management Library (CML), and Access Control Library (ACL). Each SMP library has its own API documentation files. Refer to the API documents for questions or more detailed information on how to use each library.

2. Preliminary Configuration

A directory must be created that will contain all of the SMP related software (e.g. “devel.d”, which we will use in our examples). The name of this directory is not important to the build procedures described below; all projects reference external libraries and include files relative to their location. Throughout this document, this directory will be referenced as “default SMP home directory”. Note: When compiling the SMP libraries make sure all libraries, including eSNACC, are compiled in the same mode, i.e. release or debug or you will have problems linking. The release and debug versions of the libraries are named differently.

3. Required Software

The SMP libraries require a number of other software packages to be preinstalled on the system where the libraries will be compiled. Listed below are the platforms support by the SMP libraries and the software packages that must reside on those platforms before the libraries will compile. Prepare your system by downloading and building any software packages you require according to their instructions.

Base System (libraries and tools):

Windows:

Microsoft MS Visual C++ .NET 2003 or Visual C++ 6.0
eSNACC v1.7 (SMP v2.5 will not work with any other version)

UNIX:

GNU C/C++ Compiler 3.2.2 or later
GNU Configure
GNU Make
eSNACC v1.7 (SMP v2.5 will not work with any other version)

Optional Requirements (libraries and API's)

Windows: Windows Threads
UNIX: POSIX Threads
LDAP API (The SMP has been tested with the OpenLDAP and Netscape LDAP APIs)
www.openldap.org,
http://www.mozilla.org/directory/csdk-docs/index.html
RSA BSafe 6.0 Crytologic library
Crypto++ API (Version 5.1; 5.0 is no longer supported)
www.eskimo.com/~weidai/cryptlib.html
Note: You will need the .NET 2003 patch when building with Microsoft MS Visual C++ .NET 2003
Fortezza Cryptologic Library
Microsoft CAPI - Only available under Microsoft Windows
OpenSSL Toolkit - (Version 0.9.7.e) Used by the CML for OCSP based revocation checking.
www.openssl.org/source

3.1. Enhanced SNACC ASN.1 Compiler

Download the contents of the tar'd and gzip'd distribution file containing the source code for eSNACC v1.7.

Building eSNACC v1.7 on Microsoft Visual C++ .NET 2003:

1.) Extract the contents of the zip file using WinZip to default SMP home directory. For the following steps, assume the SNACC directory was extracted to the root of a local C: drive; eg:/ C:\SNACC.

2.) From the VS .NET 2003 program, use the '**File**' '**Open Solution**' function, and select the C:\SNACC\snacc_builds.sln solution. This will open the eSNACC projects including a BuildAll project.

3.) Select the BuildAll project and right click on it. Select the '**Set As StartUp Project**' function. Set the mode to either Release or Debug (user preference). Note that this selection will either build the release or debug version of the libraries.

4.) Now that the BuildAll project has been actively set, it should appear in bold over the other projects. Select the '**Build**' item from the menu bar. If this project has already been opened and run, you may need to run the '**Clean Solution**' command. After you are sure the solution is clean, select the '**Build**' item from the menu bar again, and select the '**Build Solution**' command. This will build all of the libraries for you.

IMPORTANT: Make sure the SNACC directory is clean before you build the BuildAll project. .NET and 6.0 compiled libraries are NOT compatible and will return errors.

5.) After the build is complete, you can find all of the libraries and the eSNACC compiler executable in the C:\SMPDist directory. Note: The SMPDist directory is always created at the same level as the SNACC directory. The projects that can be

executed are compiler, snacctest, and vdatestc. The snacctest project is an extensive test of the deliverable c++-library. The vdatest project is an extensive test of the deliverable c-library. The compiler project executes the eSNACC compiler executable over a given ASN.1 file.

Building eSNACC v1.7 on Microsoft Visual C++ 6.0:

1.) Extract the contents of the zip file using WinZip to the default SMP home directory. For the following steps, assume the SNACC directory was extracted to the root of a local C: drive; eg:/ C:\SNACC.

2.) From the VS 6.0 program, use the '**File**' '**Open Workspace**' function, and select the C:\SNACC\snacc_builds.dsw workspace. This will open the eSNACC projects including a BuildAll project.

3.) Select the BuildAll project and right click on it. Select the '**Set As Active Project**' function. Set the mode to either Win32 Release or Win32 Debug (user preference). Note that this selection will either build the release or debug version of the libraries.

4.) Now that the BuildAll project has been actively set, it should appear in bold over the other projects. Select the '**Build**' item from the menu bar. If this project has already been opened and run, you may need to run the '**Clean**' command. After you are sure the project is clean, select the '**Build**' item from the menu bar again, and select the '**Build**' command. This will build all of the libraries for you.

   IMPORTANT: Make sure the SNACC directory is clean before you build the BuildAll project. .NET and 6.0 compiled libraries are NOT compatible and will return errors.

5.) After the build is complete, you can find all of the libraries and the eSNACC compiler executable in the C:\SMPDist directory. Note: The SMPDist directory is always created at the same level as the SNACC directory. The projects that can be executed are compiler, snacctest, and vdatestc. The snacctest project is an extensive test of the deliverable c++-library. The vdatest project is an extensive test of the deliverable c-library. The compiler project executes the eSNACC compiler executable over a given ASN.1 file.

Common Microsoft Windows Building Problems and Solutions:

1.) Workspace/Solution will not load or no projects can be opened.

   a. Make sure that you have used WinZip or an equivalent extraction program to extract eSNACC. Often when using tools other than WinZip the extraction tool does not recognize the carriage return/line feed at the end of the workspace and project files. This will cause them to be extracted improperly and they will not be able to be opened.

2.) Build command is returning multiple errors referencing library functions.

   a. Make sure that the workspace/solution is cleaned before the build is executed. Chances are that you are referencing an out of date or wrong library.

3.) SMPDist could not be created or could not be found.

a. Make sure that the permissions at the SNACC level of the directory are set for editing. SMPDist needs to be built automatically by the build process.

Building eSNACC v1.7 on Unix platforms using gcc 3.2.2 and 3.3.2:

1.) Use the gunzip tool on the gz'd eSNACC v1.7 tarball.
2.) Using the tar command, extract the SNACC directory from the tarball to the default SMP home directory.
3.) After extraction, execute the '**configure**' script found at the root level of the SNACC directory.
4.) After a successful execution of the '**configure**' script, run the '**make**' command. This will build the eSNACC executable as well as the c/c++ libraries.
5.) After a successful '**make**', run the '**make install**' command. The result of the build and make install should be a set of include files in the ./SMPDist/include/esnacc/ … for both 'C' and 'C++', and the SNACC compiler binaries in the ./SMPDist/bin/ directories. You are now ready to build the SMP libraries.

Unix Configuration Options:

The eSNACC configure process allows you to specify different configuration options, in order to build the eSNACC libraries properly. The configuration options can either be passed as flags (--enable, --with) or via environment variables set before the configure shell is started.

The options recognized by the eSNACC configure process are:

Option:
--prefix
Associated Environment variable: None

- This option is used to specify a different install directory other than the default (/usr/local). This option is used to place the include files in [prefix]/include/smp and the library files in [prefix]/lib. The installation directory MUST be structured as a normal install directory would be. It MUST contain a bin, lib, and include subdirectories, such as /usr/local.

  Example: --prefix=/home/george/application

Option:
--enable-debug
Associated Environment variable: None

- This option enables debug libraries to be built. These libraries will be named with a "_d" appended to the name to denote a debug library. For example, a non-debug library is called <library>.so and the debug library is called <library>_d.so.

Option:
--disable-threads
or
--enable-threads=no
Associated Environment variable: None

- This option disables threads when compiling the SMP libraries. By default, threads are enabled. Note: disabling threads on an operating system that supports threads will make the code thread *unsafe*.

3.1.1. Dependencies

eSNACC expects the following executables to be present:

- OPTIONAL: Flex
- OPTIONAL: Bison

If you do not have these utilities, the default distribution places a MS Windows compatible source file in the appropriate directories. They are only important if you change any *.l or *.y lex/yacc source files.

These files are located in the following directories:

./SNACC/compiler/core/lex-asn1.c
./SNACC/compiler/core/parse-asn1.h
./SNACC/compiler/core/parse-asn1.c

3.1.2. Enhanced SNACC Compiler Notes

The eSNACC compiler is placed in the following directory:

./SMPDist/bin/

3.2. Crypto++ Library

Download the Crypto++ freeware library from http://www.eskimo.com/~weidai/cryptlib.html. SMP version 2.5 was tested with Crypto++ version is 5.1. Do not attempt to use SMP with any other versions.

On windows platforms, extract the library to a subdirectory named cryptopp (e.g. ./devel.d/cryptopp). It is expected to be at the same directory level as the “smp” directory in the default SMP home directory. On UNIX platforms you need to build and install Crypto++ according to Crypto++’s build instructions. If the library was not installed to the default location you will need to use the **--with-cryptoppincdir** and **--with-cryptopplibdir** options to configure as documented in Appendix A.

After SMP is built, the header and library files are moved to ../../SMPDist/Algs/Crypto++. This is where the SFL Free3 CTIL expects the source includes and libraries to reside.

3.3. LDAP Library

LDAP Note: If you wish to support remote retrieval of certificates and CRLs, a Lightweight Directory Access Protocol (LDAP, RFC 1777) library, compatible with Netscape's LDAP SDK, or OpenLDAP is required. The Netscape LDAP SDK can be downloaded from the Netscape site for developers at <http://www.mozilla.org/directory/csdk-docs/index.html> or OpenLDAP at <http://www.openldap.org>

3.4. OpenSSL Library

The OpenSSL library is needed if you want to use OCSP based revocation checking. You can download OpenSSL source code from www.openssl.org/source.

On windows platforms, extract the library to a subdirectory name openssl.0.9.7e (e.g. ./devel.d/openssl.0.9.7e). It is expected to be at the same directory level as the “smp” directory in the default SMP home directory. When the library is extracted it will be placed in a directory called “openssl.0.9.7e”. You will need to change the name of this directory to “openssl” so it can be used with the SMP libraries. On UNIX platforms you need to build and install OpenSSL according to OpenSSL’s build instructions. If the library was not installed to the default location you will need to use the **--with-openssldir** option to configure as documented in Appendix A.

3.5. BSAFE Library

The Bsafe Library is needed if you want to use the SFL’s RSA CTIL. You can contact RSA Security at www.rsasecurity.com for the BSafe crypto library.

The BSafe60 library is distributed as object code, and as such, must be installed manually by placing the distribution files in the appropriate directories as follows:

On the windows platform, the Bsafe60 include files must be placed in the following directory:

../SMPDist/Algs/Bsafe60/Library/include

The library file libbsafe.lib, must to be placed in the following directory:

../SMPDist/Algs/Bsafe60/Library/lib

The source file tstdlib.c, must to be placed in the following directory:

../SMPDist/Algs/Bsafe60/Library/source

On UNIX platforms, the Bsafe60 distribution can be located by the **configure** script in two different ways. By default, **configure** looks for the files in the following locations:

The Bsafe60 include files in:

/usr/local/include

The library file, libbsafe.a, in:

/usr/local/lib

The source file, tstdlib.c, in:

/usr/local/src/bsafe

If you do not want to place the Bsafe60 files in the default locations, the the Bsafe60 files can be located is by running the SMP configure script with the **--with-bsafelibdir**, **--with-bsafeincdir**, and **--with-bsafesrcdir** options prior to building the SMP libraries as described in Appendix A.

3.6. Fortezza Library

The Fortezza Library is needed if you want to use the Fortezza CTIL. The Fortezza CTIL requires the TSSP32.lib/dll library, available only from the U.S. Government. The SFL expects the TSSP32.lib file to be in the ../SMPDist/lib directory.

3.7. Spex Library

The Spex Library is needed if you want to use the Spex CTIL. SPYRUS can be contacted at www.spyrus.com.

4. Distribution Directory Layout

The SMPDist distribution directory was created to avoid confusion when linking applications with the SMP libraries. It provides a common set of directories with a constant name for all project linkages. It also provides a separate location to contain SMP include and library files without the source code. This allows applications to point to a single location when linking with the SMP libraries and including the SMP header files. On the Windows platform, each SMP workspace is configured to distribute its files into the appropriate location in the distribution directory. On UNIX platforms, each Makefile performs this same functionality.

In order to use our projects, the Windows environment or UNIX environment MUST point the PATH to the ./SMPDist/Bin directory to execute certain components (i.e. the Enhanced SNACC compiler executable at this time).

The following list demonstrates the layout of the distribution sub-directory:

SMPDist/Algs:
SMPDist/Algs/Crypto++
SMPDist/Algs/Fortezza
SMPDist /Algs/Spex
SMPDist /Algs/BSafe60

SMPDist/bin:	# esnacc executables.

SMPDist/include:

SMPDist /include/esnacc/c	# eSNACC C lib headers
SMPDist /include/esnacc/c++	# eSNACC C++ lib headers
SMPDist/include/pkcs11	# pkcs11 headers
SMPDist/include/smp	# ALL smp include files
SMPDist/include/smp/Modules	# ASN.1 files

SMPDist/lib:	# ALL smp library files

5. SMP Libraries

5.1. S/MIME Freeware Library (SFL) libraries

The SFL consists of the following libaries:

- libCert and libsm, SFL High-Level libraries which implement all CMS and ESS operations. (requires CML)
- libCtilMgr, generic interface to external crypto token libraries
- CTILs (optional token dependent cryptographic interface libraries)

The CTILs provide the actual cryptographic functionality. Each CTIL links to a different user-provided library. The CTILs are described below:

sm_free3	based on the freeware Crypto++ 5.1 library (includes and libraries in ./SMPDist/Algs/crypto++/)
sm_rsa	based on the RSA Bsafe 6.0 library (includes and libraries in ./SMPDist/Algs/Bsafe60/Library/)
sm_spex	based on the Litronics SPEX/Fortezza library (includes and libraries in ./SMPDist/Algs/spex/)
sm_fort	based on Fortezza (hardware/software) (includes and libraries in ./SMPDist/Algs/fortezza/)
sm_pkcs11	based on the PKCS11 specification, various implementations

(nothing necessary to build; must have application specific
PKCS11 DLL library to load dynamically).

5.2. Certificate Management Library (CML)

The CML consists of five separate libraries:

- cmapi, the main CM library
- crlapi, the certificate revocation list server library (optional, requires srlapi)
- ocspapi, the OCSP revocation status checking library (optional, requires ocsplib_mod)
- srlapi, the Storage and Retrieval library (optional, requires LDAP)
- cmlasn, the CML ASN.1 library for encoding and decoding X.509 certificates and other ASN.1-encoded objects. (requires ESNACC)

In addition to those five libraries, the CML calls functions in a PKCS #11 library (i.e., Cryptoki) or Microsoft's CAPI library (only on Windows) to perform the required cryptographic functions. The SMP distribution includes a PKCS #11 library (pkcs11_cryptopp_dll) which can be used with the CML for cryptographic support. This PKCS #11 library requires Crypto++ version 5.1. The CML also requires the eSNACC C++ ASN.1 library in order to encode and decode ASN.1 objects. These libraries and their header files must be present in order to build any one of the libraries in the CML.

5.3. Access Control Library (ACL)

The ACL will be built when the SMP build is performed. See the directions below to build the SMP libraries. See the ACL documentation for more information about the ACL.

6. SMP Dependencies

By default, SMP expects the following libraries and executables to be present:

- eSNACC v1.7
- Crypto++ Library (for SFL's Free3 CTIL and CML's optional PKCS 11 library)
- OpenSSL (for CML OCSP based revocation status checking)

Each of these libraries are described in this document; they must be retrieved and setup before continuing.

7. Source Code for the eSNACC and SMP Libraries

Follow the following steps to retrieve the source code for the eSNACC and SMP Libraries:

1. Download the contents of the tar'd and gzip'd distribution file containing the source code for the eSNACC v1.7 C/C++ library and compiler.

2. Download the tar'd and gzip'd distribution file containing the source code for the Secure Message Protocol (SMP) v2.5 libraries (including the CML).

3. If using BAE System's PKCS #11 library for cryptographic support, download the Crypto++ version 5.1 source code. In addition, if using the Visual C++ .NET 2003 compiler, download the Crypto++ patch for that specific compiler.

4. If using BAE System's OCSP revocation status checking library, download the OpenSSL 0.9.7e source code.

8. Building the SMP Libraries

After you have downloaded the source code for the SMP libraries as described above, you are ready to build the SMP libraries. If you are building the SMP libraries on a Microsoft Windows platform using the Visual C++ 6.0 compiler read section 8.1 below. . If you are building the SMP libraries on a Microsoft Windows platform using the Visual C++ .NET 2003 compiler read section 8.28.1 below. If you are building the SMP libraries on a UNIX platform using the GCC compiler read section 8.3 below.

8.1. Building SMP on Windows using Visual C++® 6.0

The following steps build the SMP libraries on Windows using Visual C++®6.0:

1. Extract the contents of the Crypto++ and OpenSLL distribution files if necessary.

    a. If using BAE System's PKCS #11 library for cryptographic support, extract the Crypto++ source code in a directory named "cryptopp" in the same directory where the SMP source code is extracted.
    b. If using BAE System's OCSP revocation status checking library, extract the OpenSSL source code in the same directory where the SMP source code will be extracted and rename the top-level directory from "openssl-0.9.7e" to "openssl".

2. Build the eSNACC 1.7 'BuildAll' project using the steps in section 3.1.
3. Open the smp.dsw workspace.
4. If not using the sm_FortezzaDLL, remove the sm_FortezzaDLL project from the BuildAll project.
5. If not using the sm_pkcs11DLL, remove the sm_pkcs11DLL project from the BuildAll project.
6. If not using the sm_pkcs11Free3DLL, remove the sm_pkcs11Free3DLL project from the BuildAll project.
7. If not using the sm_rsaDLL, remove the sm_rsaDLL project from the BuildAll project.

8. If not using the sm_spexDLL, remove the sm_spexDLL project from the BuildAll project.
9. If not using the OCSP revocation status checking library, remove the ocsp_dll and ocsplib_mod projects from the BuildAll project.
10. Build the "BuildAll" project.

8.2. Building SMP on Windows using Visual C++® .NET 2003

The following steps build the SMP libraries on Windows using Visual C++® .NET 2003:

1. Extract the contents of the Crypto++ and OpenSLL distribution files if necessary.

    a. If using BAE System's PKCS #11 library for cryptographic support, extract the Crypto++ source code in a directory named "cryptopp" in the same directory where the SMP source code is extracted.
    b. If using BAE System's OCSP revocation status checking library, extract the OpenSSL source code in the same directory where the SMP source code will be extracted and rename the top-level directory from "openssl-0.9.7e" to "openssl".

2. Build the eSNACC 1.7 'BuildAll' project using the steps in section 3.1.
3. Open the smp.sln solution.
4. If not using the sm_FortezzaDLL, remove the sm_FortezzaDLL project from the BuildAll project.
5. If not using the sm_pkcs11DLL, remove the sm_pkcs11DLL project from the BuildAll project.
6. If not using the sm_pkcs11Free3DLL, remove the sm_pkcs11Free3DLL project from the BuildAll project.
7. If not using the sm_rsaDLL, remove the sm_rsaDLL project from the BuildAll project.
8. If not using the sm_spexDLL, remove the sm_spexDLL project from the BuildAll project.
9. If not using the OCSP revocation status checking library, remove the ocsp_dll and ocsplib_mod projects from the BuildAll project.
10. Build the "BuildAll" project.

8.3. Building SMP on UNIX (Solaris and Linux)

The following steps build the SMP libraries on UNIX (Solaris or Linux) using GNU C/C++ Compiler 3.2.2 or later:

1. Build the eSNACC 1.7 project using the steps in section 3.1.

2. Extract the contents of the Crypto++ and OpenSLL distribution files if necessary.

    a. If using BAE System's PKCS #11 library for cryptographic support, extract and build the Crypto++ source code. If you do not copy the Crypto++ library and header files to their default locations, you will need to specify the --with-

cryptoppincdir and --with-cryptopplibdir options when running configure (see Configuration Items Appendix A below).

b. If using BAE System's OCSP revocation status checking library, extract and build the OpenSSL source code. If you do install the OpenSSL library and header files to their default locations, you will need to specify the -- with-openssldir option when running configure (see Configuration Items Appendix A below).

3. Unpack the SMP distribution:
   tar xfz smp2.5.tar.gz
   cd smp

4. Build a configure script to set the configure options. See Appendix A for SMP options. List additional options by typing:
   ./configure --help

5. Run the configure script to automatically detect the appropriate settings and produce the Makefiles for the SMP libraries. You may need to specify options (see Appendix A) and/or environment variables to the configure script to obtain the desired results for your particular environment. The configure script is executed by typing:
   `[environment settings] ./configure [options]`

6. Build the libraries by typing:
   make

7. Test the SMP libraries by typing:
   make check

8.3.1. Unix (Linux/Solaris) Post Installation

Add the SMPDist/lib directory to the LD_LIBRARY_PATH environment variable so that the SMP shared libraries can be dynamically loaded at application run time. For example, if the "default SMP home directory" is /usr/local, then add the /usr/local/SMPDist/lib to the LD_LIBRARY_PATH environment variable using the appropriate commands for your shell. This will assure that SMP shared libraries are properly located at run time.

9. SMP_Check Test

To verify that the SMP libraries and their dependencies were built correctly, run the SMP_Check test. On UNIX platforms, make sure the LD_LIBRARY_PATH has been set before running SMP_Check to assure proper loading of the SMP shared libraries. See section 8.3.1.

To run the SMP_Check test on the windows platform make sure that the SMP_Check project was compiled successfully. Set the SMP_Check project as active. Run the SMP_Check test.

To run the SMP_Check test on the LINUX/Solaris platform run the following commands at the command prompt in the smp directory:

make check

This command will compile the SMP_Check if necessary and run the test.

10. SMP Release Test Platforms

The SMP libraries including the CML and eSNACC v1.7 were tested on Windows using Microsoft Visual C++ 6.0 and .NET 2003, on Solaris 8 using GCC 3.3.2 and on Red Hat Linux release 9 using GCC version 3.2.2.

Appendix A Unix (Linux/Solaris) Configure Options

The SMP configure process allows you to specify different configuration options, in order to build the SMP libraries properly. The configuration options can either be passed to configure as an option (--enable, --with) or via environment variables set before the configure shell is started.

The options recognized by the SMP configure process are:

Option:
--prefix
Associated Environment variable: None

- This option is used to specify a different install directory other than the default (/usr/local). This option is used to place the include files in [prefix]/include/smp and the library files in [prefix]/lib. The installation directory MUST be structured as a normal install directory would be. It MUST contain a bin, lib, and include subdirectories, such as /usr/local.

  Example: --prefix=/home/george/application

Option:
--enable-debug
Associated Environment variable: None

- This option enables debug libraries to be built. These libraries will be named with a "_d" appended to the name to denote a debug library. For example, the non-debug SRL library is called libsrlapi.so and the debug library is called libsrlapi_d.so.

Option:
--disable-threads
or
--enable-threads=no
Associated Environment variable: None

- This option disables threads when compiling the SMP libraries. By default, threads are enabled. Note: disabling threads on a operating system that supports threads will make the code thread *unsafe*.

Option:
--with-smpdistdir
Associated Environment variable: SMPDISTDIR

- This option will set the default SMPDist directory. The SMPDist directory is used by the configure/make process as a staging directory for the install process and as an internal library and include directory storage area for the SMP. The default SMPDist directory will be one directory level up from your SMP directory. The SMPDist directory MUST

be structured to resemble /usr/local (must have subdirectories lib, and include) These directories will be created if they don't exist.

Example: --with-smpdistdir=/home/homer/smp/SMPDist

Option:
--with-cryptoppincdir
Associated Environment variable: CRYPTOPPINCDIR

- This option will set the default Crypto++ header file directory to the directory specified instead of the default /usr/local/include/cryptopp.

Example: --with-cryptoppdir=/home/george/cryptopp

Option:
--with-cryptopplibdir
Associated Environment variable: CRYPTOPPLIBDIR

- This option will set the default Crypto++ library file directory to the directory specified instead of the default /usr/local/lib.

Example: --with-cryptoppdir=/home/george/cryptopp

Option:
--with-openssldir
Associated Environment variable: OPENSSLDIR

- This option will set the default OpenSSL directory to the directory specified instead of the /usr/local/ssl

Example: -- with-openssldir=/home/george/ssl

Option:
--with-snaccdir
Associated Environment variable: SNACCDIR

- The --with-snaccdirexec will set the default SNACC library and include directory root (the default is /usr/local).

Example: --with_snaccdir=/home/george/snacc

Option:
--with-snaccdirexec
Associated Environment variable: SNACCDIREXEC

- The --with-snaccdirexec will set the default SNACC executable to the executable passed in (the default is /usr/local/bin/snacc).

Example: --with_snaccdirexec=/home/george/bin/snacc

Option:

--enable-static

Associated Environment variable: None

- This option builds static libraries, instead of shared libraries. Note: Shared and static libraries cannot be built together. This option is not recommended.

Option:

--enable-onlycml

Associated Environment variable: None

- This option configures the make files to only build the CML libraries, and any SMP library the CML uses. Using this flag will not build all of the SMP, including SMP_Check.

Option:

--with-bsafelibdir

Associated Environment variable: BSAFELIBDIR

- The --with-bsafelibdir will set the default Bsafe Library directory to the directory specified instead of /usr/local/lib. To specify another directory enter:

  Example: --with-bsafelibdir=/usr/foo/bsafe

Option:

--with-bsafeincdir

Associated Environment variable: BSAFEINCDIR

- The --with-bsafeincdir will set the default Bsafe Include directory to the directory specified instead of /usr/local/include/bsafe. To specify another directory enter:

  Example: --with-bsafeincdir=/usr/foo/bsafe

Option:

--with-bsafesrcdir

Associated Environment variable: BSAFESRCDIR

- The --with-bsafesrcdir will set the default Bsafe source directory to the directory specified instead of /usr/local/src/bsafe. To specify another directory enter:

  Example: --with-bsafesrcdir=/usr/foo/bsafe

Option:

--with-fortezzadir

Associated Environment variable: FORTEZZADIR

- This option will set the default Fortezza directory to the directory specified instead of /usr/local/lib and /usr/local/include. To specify a different directory enter:

  Example: --with-fortezzadir=/usr/foo/Fort