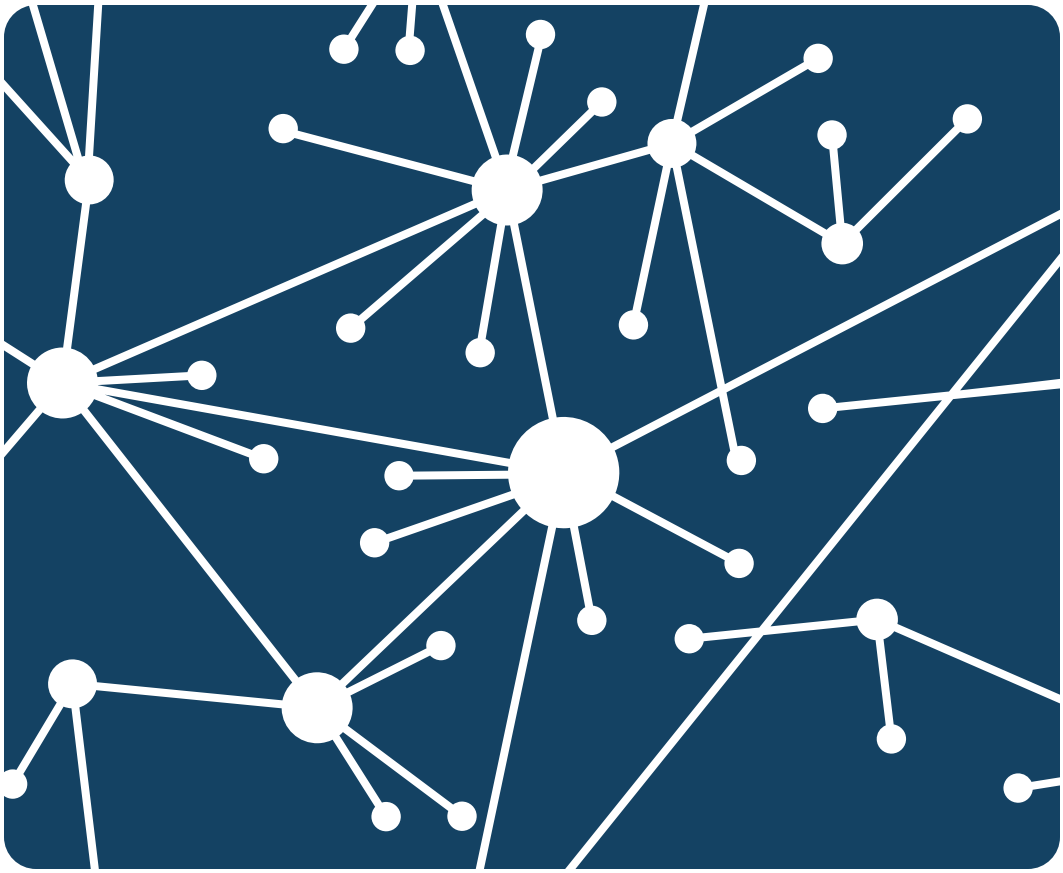
communications

Security Policy for the CoCo Crypto Module v1.0

Using CoCo's DLL of cryptographic functions for compliance with FIPS 140-2 Level 1



Usage Policy

Security Policy for the CoCo Crypto Module v1.0

Using CoCo's DLL of cryptographic functions for compliance with FIPS 140-2 Level 1

Document Revision 122

28 January 2008

CoCo Communications Corporation

www.cococorp.com

999 3rd Ave, Suite 3700

Seattle, WA 98104

Phone: 206-284-9387

Fax: 206-770-6461

Toll free: 866-657-COCO

Table of Contents

I. Summary

This document contains the security policy for the CoCo Crypto Module, a dynamically-linked library (DLL) and corresponding C++ -oriented Software Development Kit (SDK) intended for use in other CoCo Communications products in order to facilitate compliance with FIPS 140-2 requirements. The DLL contains cryptographic algorithms implemented in a manner compliant with FIPS 140-2 Level 1.

1. Document References

This document is part of a set of documents that collectively comprise the usage guidelines for the cocoCrypto module. The following list enumerates the documents in this collection.

- Security Policy for the CoCo Crypto Module v1.0
- Finite State Model for the CoCo Crypto Module v1.0
- API Reference for the CoCo Crypto Module v1.0
- Configuration Management List for the CoCo Crypto Module v1.0 (proprietary document; distribution is limited exclusively to individuals and agencies involved in the process of acquiring FIPS 140-2 certification for this module)

2. Documentation Conventions

Within the scope of documentation and reference for the CoCo Crypto Module, the CoCo Crypto Module may be abbreviated as “CoCo Crypto”, “cocoCrypto”, or simply as “the Module” or “the module”. As a general rule, the camel-cased moniker “cocoCrypto” is used to refer to the module when discussing its API, build configuration, or component files (particularly the DLL binary file containing the module’s implemented algorithms), whereas other abbreviations are typically used when discussing the module’s behavior and functionality.

II. CoCo Crypto Module Overview

CoCo Communications Corporation is a Seattle-based technology company specializing in next-generation communications software. The company's operational focus is a patent-pending cryptographic mesh protocol that solves many of the problems currently plaguing the communications industry. The initial application of this breakthrough capability is helping to solve the interoperability, availability, and security challenges facing our nation's first responders.

When called to dangerous and potentially hostile environments, first response teams need to know that they can communicate in a secure and reliable manner – they depend on their equipment to keep them connected with local, state, and federal agencies. However, they also must keep sensitive tactical information from reaching potentially harmful recipients, whether that be terrorists or combatant forces, or simply a panicked civilian populace.

CoCo Communications is dedicated to providing emergency response personnel with equipment they can trust. To this end, all CoCo products rely on a routing protocol that has cryptographic guarantees built into its framework. CoCo software uses a dynamically-linked library containing FIPS-approved implementations of cryptographic algorithms such as encryption/decryption, signed hashing, and certificate validation. CoCo's products rely on this library for all cryptographic operations. The library is referred to as the CoCo Crypto Module and is the topic of this security policy document.

The CoCo Crypto Module is an integral component of CoCo Communications' product offerings and its certification by the CMVP establishes its adherence to FIPS 140-2 Level 1 guidelines. The Module is critical to providing CoCo's customers the secure, reliable, and trustworthy communication systems on which they depend.

1. Supported Algorithms

a. Approved Algorithms

The Module implements the following cryptographic algorithms. The module's implementations of these algorithms have been tested and validated for FIPS 140-2 Level 1 compliance by Atlan Laboratories.

- AES
  - Supported modes: ECB, CFB128
  - Supported functions: Encrypt, Decrypt
  - Supported key sizes (in bits): 128, 192, 256
- SHA-1 (byte-oriented)
- HMAC SHA-1
  - MAC sizes (in bytes): 10, 12, 16, 20
- Pseudorandom number generator using 2-Key Triple-DES
  - Adheres to ANSIX9.31 standard, Appendix A.2.4
  - Usage
    - General-purpose random number generator
    - All FIPS Approved key generation
- DSA
  - Implemented according to FIPS 186-2 standard
  - Features Supported
    - PQG Generation
    - Key Pair Generation
    - Signature Generation
    - Signature Verification
  - Modulus Sizes Supported (in bits): 1024, non-compliant less than 1024 bits. The user should not use non-compliant moduli less than 1024 bits in the Approved mode.

b. Non-Approved Algorithms

In addition to the validated algorithms listed above, the module also contains implementations of algorithms without FIPS validation certificates. These algorithms are listed as follows.

- Diffie-Hellman Primitives

    - Features Supported
        - Key Pair Generation
        - Shared Secret Computation
    - Strength: The Diffie-Hellman key establishment primitives calculate shared secrets providing between 80 and 256 bits of encryption strength; non-compliant less than 80 bits of encryption strength. Diffie-Hellman parameters less than 1024 bits in length, which can generate shared secrets with less than 80 bits of encryption strength, should not be using in the Approved mode.
- SSLeay RNG
    - Usage: Seeding the Approved PRNG
    - Continuous RNG Test: Loop detection by comparison of each generated value with previous

2. Supported Platforms

The module has received explicit testing and verification by Atlan Laboratories for FIPS 140-2 Level 1 compliance on the following platforms.

- Microsoft Windows XP Pro Service Pack 2 on Intel x86 or compatible chipsets.
- Debian GNU/Linux 4.0 (Etch) on Intel x86 or compatible chipsets.

CoCo Communications also affirms the CoCo Crypto Module on additional platforms, listed below. When building the module for these additional platforms, CoCo Communications uses the same toolset and source code as used for the explicitly approved platforms listed above. As per FIPS 140-2 IG G.5, the module maintains Level 1 compliance on these platforms.

- Microsoft Windows 2003 Server on Intel x86 or compatible chipsets.
- Microsoft Windows Mobile 5.0 Pocket PC Edition on an ARM chipset.
- Microsoft Windows Mobile 6.0 Pocket PC Edition on an ARM chipset.
- Debian GNU/Linux 4.0 (Etch) on a MIPS chipset.
- Debian GNU/Linux 4.0 (Etch) on an ARM chipset.

3. Product Components

The module, being a software product, consists of a set of files. This section enumerates the files that comprise the CoCo Crypto Module.

a. Core Components

This section lists the necessary and sufficient components that need to be installed on a host computer in order to make use of the cryptographic capabilities of the module.

i. Dynamically-Linked Library File (DLL)

This binary file contains the compiled implementations of the methods and objects exposed and supported by the module. It is the functional core of the module, and was the primary focus of testing during the FIPS 140-2 Level 1 certification process.

On Windows platforms, this file is called `"cocoCrypto.dll"`.

On Debian platforms, this file is called `"libcocoCrypto.so"`.

ii. Signature File

When the CoCo Crypto Module enters FIPS mode, it performs a test to verify the integrity of the DLL. This test takes the form of computing the signed HMAC-SHA1 hash of the DLL and comparing it against a known value. This known value resides in a small standalone file.

On all supported platforms, this file is called `"cocoCrypto.sig"`.

b. Software Development Kit (SDK)

When CoCo Communications provides the CoCo Crypto Module to third-party or internal customers, the module comes with files that software developers might find useful for authoring applications that leverage the module's capabilities.

These files are not necessary for the operation of the CoCo Crypto Module by the end user of the application, and are not included in application packages that aren't intended for software developers. The SDK performs no cryptographic operations of its own and did not receive validation as part of the CoCo Crypto Module. It exists solely to provide software developers with a mechanism which to access the functions in the DLL.

i. Import Library

When building C++ applications for the supported Microsoft Windows platforms, a software developer generally needs to link against an import library, a small static library containing stubs for the methods in a DLL, in order to resolve references at compile time.

On all Windows-related platforms, this file is called `"cocoCrypto.lib"`.

Import libraries are not necessary for Debian Linux and related platforms, and are therefore not provided.

ii. C/C++ Headers

The module's SDK includes a header file which presents the C++ functional entry points and object declarations, which together form the user interface to the module.

On all supported platforms, this file is called `"cocoCrypto.h"`.

4. Build Configuration

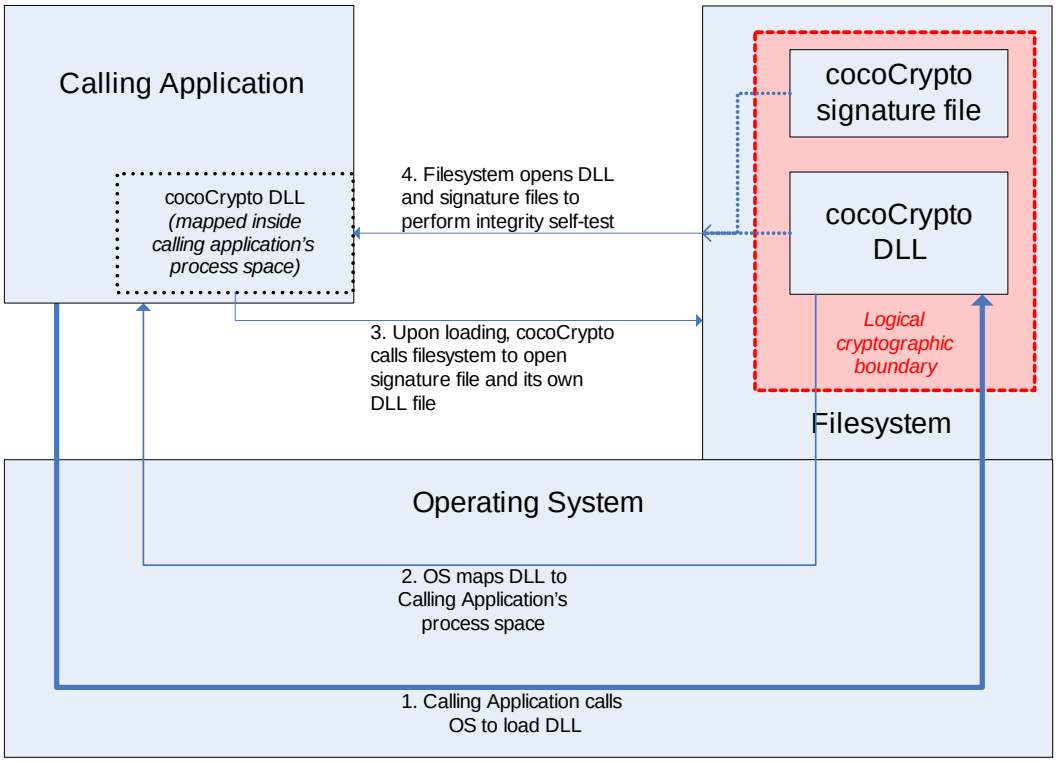
The CoCo Crypto Module was built by CoCo Communications Corporation on a set of Intel x86 or compatible machines, using Microsoft Visual Studio 2003 Enterprise Edition on Microsoft Windows XP Professional SP2 and gcc 3.4.4 on Debian (Sarge) Linux. The source code comprising the CoCo Crypto module was stored in a repository managed by Subversion (SVN) v1.4.2 (more information about SVN available at http://subversion.tigris.org/).

A full listing of the files in the SVN repository, including revision numbers at the time of FIPS submission of this module, is available in the accompanying document, *Configuration Management List for the CoCo Crypto Module v1.0* (proprietary distribution only).

5. Module Architecture

CoCo Crypto is a multi-chip standalone module. As a software module, CoCo Crypto's physical cryptographic boundary is the computer case of the machine on which the cocoCrypto DLL is installed. Its logical boundary is the two files that comprise the module – the cocoCrypto DLL and the signature file, as described in this document section II.3.a, *Core Components*.

The following illustration is a logical diagram that describes the interaction between each of the module's components (DLL and signature file), the operating system, and a calling application.



III. Roles and Authentication

This section defines and discusses the various roles of the intended users of the CoCo Crypto Module.

1. Role Definitions

a. User

The module, as it is a software library, defines its "user" to be the application process which invokes the loading and run-time linking of the cocoCrypto DLL.

For purposes of classification, when referred to as a "User", such an application process runs with restricted permissions on the host computer. For example, such a process would be launched by a generic account on a Linux-based machine, or a guest or restricted account on a Windows-based machine.

b. Crypto Officer

The module, as it is a software library, defines its "Crypto Officer" to be the application process which invokes the loading and run-time linking of the cocoCrypto DLL.

For purposes of classification, when referred to as a "Crypto Officer", such an application process runs with elevated permissions on the host computer. For example, such a process would be launched by a root user account or privileged daemon on a Linux-based machine, or an administrator account on a Windows-based machine.

2. Restrictions and Privileges

As permitted by FIPS 140-2 Level 1 guidelines, the module enforces no functional difference between a user and a crypto officer. The user and the crypto officer both have full access to the complete set of functions in the module.

The remainder of this document will employ the term "user" to refer to either a User or a Crypto Officer, since both have the same capabilities with regard to the module. The reader should bear in mind that the term refers specifically to an application process running on a host computer, and not to a human operator.

3. Authentication

As a software module with FIPS 140-2 Level 1 certification, CoCo Crypto does not provide authentication mechanisms of its own. It depends on the operating system of the host computer to enforce the access privileges of any given process and to invoke the loading and run-time linking of the cocoCrypto DLL.

4. Multiple user concurrent access

When a process links against a dynamically-linked library, the operating system provides that process with its own copy of the library's data segment. This means that any such process has no way to alter the behavior of any other process by way of the DLL.

Therefore, we place no procedural restrictions on the number of concurrent users (i.e. processes) accessing the CoCo Crypto Module on a given host computer at any one time. Any practical limits on such a number come from the CPU, memory, and operating system constraints of the host computer.

IV. Secure Operation and Security Rules

In order to use CoCo Crypto securely in a manner compliant with FIPS 140-2 Level 1 requirements, a system administrator should adhere to the policies and procedures described in this section.

1. Preparing a Secure Operating Environment

a. Availability of the DLL file

The CoCo Crypto Module requires the system administrator to place the cocoCrypto DLL file in a specific location on the file system.

The module needs to perform an integrity check on the file image of the cocoCrypto DLL when entering FIPS mode. Therefore, failure to place this file in the proper location will result in termination of the calling process upon an attempt to enter FIPS mode.

The system administrator must ensure that the user's library load path specifies this location, and that no other file exists in any library load path with the same name as the cocoCrypto DLL. Failure to do so could result in a client application linking to some different DLL, which may contain malicious code.

On Windows systems, the DLL must be located in the file system under the path `C:\WINDOWS\system32\`.

On Linux systems, the DLL (shared object file) must be located in the file system under the path `/opt/coco/lib/`.

b. Availability of the signature file

To make use of the CoCo Crypto Module, the user must have permission to read the signature file, which must be in a specific location on the host computer's file system.

It is the system administrator's responsibility to ensure that the signature file exists and is accessible at the appropriate location. Failure to do so will result in termination of the calling process upon an attempt to enter FIPS mode.

On Windows systems, the signature file must be located in the files system under the path `C:\WINDOWS\System32\`.

On Linux systems, the signature file must be located in the file system under the path `/opt/coco/etc/`.

c. Running the host OS in single-user mode

Each operating system supported by the Module has a means by which to be placed into single-user mode. This subsection contains instructions on how to do so.

i. Windows XP

The user must configure the Windows XP machine to run in single user mode, ensuring that remote login is disabled and that the workstation cannot be accessed as a server.

1.

ii. Debian Linux

Debian uses a boot loader called GRUB to bootstrap the operating system. Upon starting (or restarting) the machine, GRUB will present the user with a menu of boot options (in order to see this menu, the user may need to hit the Escape key during boot, much like hitting the F8 key during booting of Windows XP). Most individual Debian systems will come configured with a "single-user mode" entry in this menu. The user should use the arrow keys to select this entry and press "Enter" to boot the system into single-user mode.

If the system does not have a "single-user mode" menu entry on the GRUB menu, the user can follow these steps in order to make one of the existing menu entries cause the system to boot into single-user mode.

1. Use the arrows to select the boot entry to modify.
2. Press *e* to edit the entry.
3. Use the arrows to go to the "*kernel*" line.
4. Press *e* to edit this entry.
5. At the end of the line add the word "*single*".
6. Press ESC to go back to the parent menu.

7. Press *b* to boot this kernel.

The kernel will begin booting as usual (except without a graphical splash screen if one is normal for that system), and the user will receive a command-line interface from which the user can log in as root.

2. Activating the module

A process activates the CoCo Crypto Module by issuing a request to the host computer's operating system to load the DLL. If the process has sufficient permission, the operating system responds to this request by loading the DLL and granting the calling process access to its memory space.

Upon the loading of the DLL, the module will automatically run a series of tests to confirm that it is functioning properly (described in further detail below). If any of these tests fail, the CoCo Crypto Module will immediately terminate the calling process. This eliminates any possibility of the user intentionally or accidentally invoking a corrupted or altered version of the module's cryptographic services.

3. Self-Tests

In compliance with FIPS 140-2 Level 1 requirements, the CoCo Crypto Module performs a series of tests on itself to assure the user of its own proper functionality.

a. Power-On Self-Tests

Upon starting, the module performs a series of self-tests, listed below. If any of these tests fail, the module will immediately terminate the calling process.

i. List of Power-On Self Tests

The list of tests that the module performs upon starting is as follows.

- **Library Load Path Test.** As described in "1. Preparing a Secure Operating Environment", the cocoCrypto DLL must be loaded from a specific location on the computer's file system in order to be compliant with proper usage guidelines. It is the system administrator's responsibility to place the file in the proper location and to configure the system's library load paths to ensure that the library is loaded properly. In order to enforce this policy from within the module itself, the CoCo Crypto Module verifies, on start-up, that it has been loaded from the proper location as per usage guidelines. This test only occurs on Windows systems.
- **Module Integrity Test.** The module computes an HMAC SHA-1 signed hash of the DLL file containing the module's executable binary code, and compares the result against a known value.

- **Known Answer Tests.** The module performs a series of Known-Answer Tests (KATs) upon startup. The following table lists these tests.

Algorithm	Known Answer Test
AES	• Encryption • Decryption
HMAC	HMAC-SHA-1
RNG	Random number generation from known initialization vector.

- **Sign-Verify Test for DSA.** The module performs a Sign-Verify Test to confirm that DSA is functioning properly. This test consists of signing and signature verification of a generated key.

ii. Inducing the Power-On Self Tests

The module automatically runs the power-on self-tests whenever the cocoCrypto DLL loads. The user can run the power-on self-tests on demand by unloading and then re-loading the DLL. The following table presents the most commonly used API for performing these tasks.

Platform	Function to load DLL	Function to unload DLL	Header file
Linux	`dlopen()`	`dlclose()`	`dlfcn.h`
Windows	`LoadLibrary()`	`FreeLibrary()`	`windows.h`

b. Conditional Continuous Self-Tests

The module performs conditional continuous self-tests to verify the correctness of certain mathematical operations during the course of normal operation. Failure of any self-test at any time will cause the module to immediately terminate the calling process. The table below lists these tests.

Algorithm	Operation	Conditional Continuous Self-Test	Exposed function(s) that induce(s) self-test
DSA	Key Generation	Pairwise consistency (signing and signature verification)	`DSA_generate_key()`

RNG	Pseudorandom number generation	Repetition test (FIPS 140-2 §4.9.2)	`DH_generate_key()`
			`DSA_generate_key()`
			`DSA_generate_parameters()`

c. Checking for Self-Test Failure

The module causes the calling process to immediately terminate upon a self-test failure. When it does so, the process's exit code indicates the general reason for the termination.

Reason for Process Termination	**Exit code**	**Notes**
Normal (Clean) Exit	0	The developer of the calling application should adhere to software design best practices and consistently use an exit code of 0 to indicate clean exit.
Failure of Power-On Self-Test	Decimal: -1 (255 in one-byte two's complement) Hex: 0xFF	The developer of the calling application must refrain from using an exit code of -1, so as to ensure that this code reliably and consistently indicates failure of a power-on self-tests.
Failure of Conditional Continuous Self-Test	Decimal: -3 (253 in one-byte two's complement) Hex: 0xFD	The developer of the calling application must refrain from using an exit code of -3, so as to ensure that this code reliably and consistently indicates failure of a conditional continuous self-test.

Because the module terminates the calling process upon error, checking for self-test failure cannot be done from within the calling application, and must be performed as an out-of-band operation. Typically, the user of a computer can check an application's return code by launching that application from a command-line interface and subsequently echoing the return code. On Linux systems, the user can echo the most recently run application's return code with the command:

```
echo $?
```

On Windows systems, the same effect can be achieved with the command:

```
echo %ERRORLEVEL%
```

Note that, depending on the configuration of the user's system, the command-line interface will print the resultant error code as either a signed decimal integer with a negative value (i.e. -1 or -3), or as an unsigned integer in one-byte two's complement arithmetic (i.e. 255 or 253, respectively).

4. Physical Security

As a software module, CoCo Crypto relies on the host system for all forms of physical security. The system administrator must ensure that the host system is built with production grade components and incorporates safeguards against physical tampering.

5. Ports and Interfaces

As a software module, CoCo Crypto's interfaces are defined by the functions exposed by the cocoCrypto DLL, as listed in Appendix A of this document and described in the accompanying *API Reference* document. (Note that only those functions listed under the *Primary Functions* subsection in Appendix A are approved for secure operation.)

When mapping the module's API to the defined FIPS interfaces, it is important to recognize that any one function can have aspects of functionality that logically map to two or more interface categories. For this reason, the appropriate mapping occurs not only at the level of the functions themselves, but also at the arguments and return values of those functions. This section elaborates on the exact means by which the CoCo Crypto API maps to the FIPS interfaces.

a. Data Input Interface

The Data Input Interface consists of all function arguments that pass data into the function. This data can be passed in through a variety of mechanisms, such as by value, by reference, by pointer to a memory buffer, and so on.

b. Data Output Interface

The Data Output Interface consists of all variables and function arguments that enable a function to pass out processed data. This data can be passed out of a function through a variety of mechanisms, such as by return value, by setting the value of a return argument, by populating a memory buffer, and so on.

c. Control Input Interface

The Control Input Interface consists of all function calls (and corresponding arguments) that change or affect the overall operational state of the module.

d. Status Output Interface

The Status Output Interface consists of all variables and function arguments that enable a function to provide the user with information about the operational status of the module. This data can be passed out through a variety of mechanisms, such as by return value, by setting the value of a return argument, by populating a memory buffer, and so on. This interface would include, for example, the return values of functions that return status codes indicating success or failure of an operation.

V. Using CoCo Crypto functions and objects

This section describes usage policies for the functions and objects exposed by the header files in the CoCo Crypto Module and implemented in the cocoCrypto DLL.

1. Usage Requirements

To ensure secure operation, when accessing the module, the user must only call those functions and instantiate those objects that have public definitions in the cocoCrypto header files.

2. Usage restrictions by role and API section

Both the User and the Crypto Officer have equal access to all services offered by the module. As a software module, CoCo Crypto's services are defined as the functions exposed by the cocoCrypto DLL. Please refer to Appendix A for a full listing of every function offered by the module.

The user may wish to adhere to additional usage restrictions based on the section of the API in which a function appears. The cocoCrypto API is divided into three sections.

1. Primary Functions
2. Advanced Functions
3. Abstraction Functions

The first, "Primary Functions", contains implementations of cryptographic algorithms as well as support functions necessary to invoke those implementations. This section is of predominant interest to a user of CoCo Crypto who wishes to leverage the library's cryptographic capabilities, and therefore is the section that received focus during the FIPS validation process. In order to ensure FIPS compliance, the user should only call functions from the "Primary Functions" section, and should avoid calling "Advanced Functions" and "Abstraction Functions" while in FIPS mode.

3. Application Programming Interface

The document *API Reference for the CoCo Crypto Module* contains calling conventions and usage instructions for every function offered by the cocoCrypto library. Please consult that document for detailed function-by-function usage information.

4. Cryptographic Key Management

a. Sensitive values used internally by the module

This section lists all keys and other cryptographically sensitive data that CoCo Crypto uses for its operation.

Name / Summary	**Integrity Check HMAC Key**	**FIPS Rand Seed Keys**	**FIPS Rand Seed**
Usage	Used to compute HMAC signed hash of the cocoCrypto DLL file for power-on self-test. When cocoCrypto DLL loads, it performs an integrity check by computing a signed hash of its DLL file and comparing against a known value.	Used in setting up and running the FIPS PRNG (a 2-Key Triple DES algorithm)	Used in seeding the FIPS PRNG
Parameter Type	HMAC-SHA1 key	Two 2-Key Triple DES keys	Plain in-memory byte array
Size	104 bits	128 bits total (64 bits per key)	64 bits
Storage	Plaintext, hard-coded inside DLL binary file	Not placed in persistent storage (volatile RAM only)	Not placed in persistent storage (volatile RAM only)
Methods permitted to User	Zeroize	Zeroize	Write, Zeroize
Methods permitted to Crypto Officer	Zeroize	Zeroize	Write, Zeroize
Methods to write value	*None*	*None*	`RAND_seed()`
Methods to read value	*None*	*None*	*None*
Methods to zeroize value	Erase the DLL file (see section V.4.c of this document)	Power down the host machine to clear RAM	Power down the host machine to clear RAM

Name / Summary	**AES Key Objects**	**DSA Key Objects**	**DH Keys Objects**
Parameter Type	AES keys	DSA keypairs	DH keypairs
Size	128-, 192-, or 256-bits	1024-bit in FIPS mode; less than 1024 bits in non-FIPS mode	1024- to 15360-bits in FIPS mode; less than 1024 bits in non-FIPS mode
Storage	Ephemerally in volatile RAM	Ephemerally in volatile RAM	Ephemerally in volatile RAM
Methods permitted to User	Zeroize	Zeroize	Zeroize
Methods permitted to Crypto Officer	Zeroize	Zeroize	Zeroize
Methods to write value	*None*	*None*	*None*
Methods to read value	*None*	*None*	*None*
Methods to zeroize value	Power down the host machine to clear RAM	Power down the host machine to clear RAM	Power down the host machine to clear RAM

b. Key-Handling Functions

This subsection lists the functions that the module provides for generating, computing, and re-formatting cryptographic keys. For details on the proper usage of each function listed in this subsection, the reader should consult the Application Programming Interface documentation.

i. Key Generation

Except for the HMAC key used in the self-test, the module does not perform persistent storage of any cryptographic keys. When a user generates or accesses a key using cocoCrypto functions, that key initially exists only in volatile memory. It is up to the user to use the key and then discard it or to store it securely as appropriate to the user's needs.

All cryptographic key generation methods offered by the cocoCrypto library are listed in Appendix A, "Key Generation" section.

ii. Key Serialization and De-serialization

In addition to computing or generating new keys, the module also provides means by which to serialize and de-serialize existing data structures containing key information.

The core functions that perform serialization and de-serialization of potentially sensitive cryptographic information are listed in Appendix A, "Serialization" section, and Appendix A, "Deserialization" section, respectively.

In addition to these core serialization and de-serialization functions, the user can also induce the serialization or the de-serialization of cryptographic keys indirectly by using the X.509 API. Functions that operate on X.509 certificates do not contain their own implementations of serialization and de-serialization, but rather rely on the functions already mentioned in the above sections of Appendix A; the exact underlying function that any given X.509-handling function invokes depends on the contents of the X.509 certificate and the kinds of keys used therein. The X.509 serialization and de-serialization functions are listed in Appendix A, "X.509 Serialization" section, and Appendix A., "X.509 Deserialization" section, respectively.

c. Zeroization

This section describes the mechanisms available to a user for discarding cryptographic keys once a user is finished with them.

i. Zeroizing Keys in Storage

The only key that the cocoCrypto module stores on disk is the HMAC key used in its integrity test.

A system administrator (or a User or Crypto Officer with write permissions on the host machine's file system granted by the host's system administrator) can zero out this HMAC key by overwriting and deleting the DLL file in which the key is embedded.

A very effective way to do this on a Linux system is to use the GNU "shred" utility. This utility is part of the GNU coreutils package, and is available on all supported versions of Linux. From a command prompt, the user can zeroize the key using the following command:

```
shred -f –z -u /opt/coco/lib/libcocoCrypto.so
```

On Windows, Microsoft offers a utility called SDelete to perform a comparable level of data elimination. The following table lists information about SDelete and how to use it for the specific task of zeroing out cocoCrypto's keys.

Platform	Windows XP
Tool Name	SDelete
Current Version	1.51
Provider	Microsoft Corporation, through Windows SysInternals
URL	http://www.microsoft.com/technet/sysinternals/Security/SDelete.mspx
Notes	This utility explicitly implements the Department of Defense clearing and sanitizing standard DOD 5220.22-M.
Usage	1. Install the SDelete executable file into a directory. 2. Open a command prompt window. 3. Change your current working directory to the one in which you installed SDelete.

	4. Execute the command: `sdelete c:\WINDOWS\system32\cocoCrypto.dll`

ii. Zeroizing Keys in Memory

The CoCo Crypto Module passes keys that it has generated or computed back to the user in the form of in-memory data structures. The user can explicitly zero out these keys with a call to the `memset()` function from the standard C library. Because in-memory data exists in volatile RAM, any such keys will also be zeroed out if a system administrator powers down the host machine.

The module also provides functions to explicitly zero out keys in memory. These functions are listed in Appendix A, "Key Zeroization and Destruction" section.

5. Pseudorandom Number Generation

Pseudorandom number generation is an important part of secure operation. Because almost every key generation algorithm relies on the system's pseudorandom number generator (PRNG), this component warrants some additional discussion within this section.

a. Strength of Key Generation Methods

The CoCo Crypto Module implements a FIPS-approved 2-key Triple DES algorithm as a PRNG for use in cryptographic key generation. The module's various key generation functions in turn call the function `RAND_bytes()` to invoke the PRNG to produce random values from which to construct keys.

The module bootstraps this PRNG using an SSLeay RNG algorithm. The SSLeay RNG generates the PRNG's two Triple DES keys and a seeding vector. By FIPS calculations, the two Triple DES keys together account for 80 bits of entropy, while the seed is 64 bits long. This means that the SSLeay RNG initializes the PRNG with a total of 144 bits of entropy. The Triple DES algorithm that drives the PRNG only produces 64 bits of output at a time before changing the seed value. Because this is less than the 144 bits of entropy with which the PRNG is initialized, CoCo Crypto's initialization mechanism for the FIPS-approved PRNG provides the PRNG with adequate strength to produce secure keys.

The SSLeay RNG, in turn, uses a continuous self-test to verify that it is producing adequately unpredictable values. This self-test consists of comparing each value output by the SSLeay RNG with the previous one; if the two prove equal, the SSLeay RNG reports an error. If such an error occurs while the CoCo Crypto Module is loading, during the process of bootstrapping the FIPS-approved PRNG, then it is treated as a fatal error; the module will fail to load, and will immediately terminate the calling application. In this manner, the user always has assurance that, upon successful entry into FIPS mode, the PRNG has adequate entropy to perform key generation.

b. Protection Against Weak Seeding

When the module enters FIPS mode (i.e. when the DLL is loaded), the SSLeay RNG algorithm bootstraps the PRNG by generating two 64-bit Triple DES keys and a 64-bit initial seeding vector. CoCo Crypto ensures that all three of these 64-bit values are unique. It compares the two keys against one another, and then compares the seed against each of the keys. If any two of these three values match, CoCo Crypto treats it as a fatal error; the module will fail to load, and will immediately terminate the calling application. In this manner, the user always has assurance that, upon successful entry into FIPS mode, the PRNG has been initialized with strong seeding parameters.

In addition, as noted in section V.4.a, the user has the ability to call the function `RAND_seed()` to write seed values to the FIPS-approved PRNG. Because the module uses a 2-key Triple DES algorithm for a PRNG, it is possible that the user may try to input a value that is equal to one of the two keys used for the Triple DES algorithm, which would weaken the PRNG's effectiveness. The module's function `RAND_seed()` performs a comparison check to safeguard against this eventuality. When the user tries to set the PRNG's seed to some value, the module performs a comparison of the requested seed value to each of the two keys. If the requested seed value matches either of the two keys, the module will not set the PRNG's seed to that value. Instead, it will zero out the memory buffer with which the user passed the requested seed value into the function, signaling to the user that an error has occurred and to try a different seed value.

VI. Service Information

If you have questions that this document does not answer to your satisfaction, please feel free to contact CoCo Communications Corporation at 1-206-284-9387, or toll-free at 1-866-657-COCO.

Appendix A: Function Catalog

This section lists all of the functions available through the CoCo Crypto Module. Each of the listed functions has a corresponding entry point in the cocoCrypto library and a declaration in the header files distributed with the module. Please refer to the document *API Reference for the CoCo Crypto Module* for more information.

Each function listed in this section (i.e. all functions in the module) can be accessed by the User, the Crypto Officer, or both. The tables in this section list each function's availability to the User (under the "Usr" column) and Crypto Officer (under the "Crp Ofc" column) roles.

For the Primary Functions in the API, this function catalog also lists the following information about a function, as appropriate. (This function catalog does not list this information for the Advanced Functions since they do not contain cryptographic implementations, nor for the Abstraction Functions since, by design, those functions can perform a number of different cryptographic operations or use a variety of encoding formats based on their arguments.)

- For functions that contain implementations of specific algorithms, the relevant algorithm is specified. This applies also to functions that generate, destroy, save, and load keys for specific algorithms. The possible values for the "Algorithm" (or "Algo") column are "AES" for AES, "DH" for Diffie-Hellman, "SHA1" for SHA-1, "HMAC" for HMAC, and "PRNG" for the 2-key Triple-DES pseudorandom number generator.
- Functions that perform serialization or de-serialization of cryptographic keys (or of data structures that can contain cryptographic keys) are specified as such, using the terms "SAVE" and "LOAD", respectively, in the "Save/Load" column.
- Functions that perform serialization or de-serialization (as mentioned in the bullet point above) are labeled with the format in which they read or write data. This will be either "PEM" for PEM encoding, "DER" for DER encoding, or "TEXT" for human-readable hexadecimal text, in the "Format" or "Fmt" column.

Primary Functions

Core Cryptographic Functions

Key Generation

Function Name	Algo	Usr	Crp Ofc
`AES_set_decrypt_key`	AES	Yes	Yes

AES_set_encrypt_key	AES	Yes	Yes
DH_check	DH	Yes	Yes
DH_compute_key	DH	Yes	Yes
DH_generate_key	DH	Yes	Yes
DH_generate_parameters	DH	Yes	Yes
DH_new	DH	Yes	Yes
DH_size	DH	Yes	Yes
DH_up_ref	DH	Yes	Yes
DSA_generate_key	DSA	Yes	Yes
DSA_generate_parameters	DSA	Yes	Yes
DSA_new	DSA	Yes	Yes
DSA_size	DSA	Yes	Yes
DSA_up_ref	DSA	Yes	Yes
HMAC_CTX_init	HMAC	Yes	Yes

Key Zeroization and Destruction

Function Name	Algo	Usr	Crp Ofc
DH_free	DH	Yes	Yes
DSA_free	DSA	Yes	Yes
HMAC_CTX_cleanup	HMAC	Yes	Yes

Cryptographic Algorithms

Function Name	Algo	Usr	Crp Ofc
AES_cfb128_encrypt	AES	Yes	Yes
AES_decrypt	AES	Yes	Yes
AES_encrypt	AES	Yes	Yes
DSA_sign	DSA	Yes	Yes
DSA_verify	DSA	Yes	Yes
HMAC_Final	HMAC	Yes	Yes
HMAC_Init_ex	HMAC	Yes	Yes
HMAC_Update	HMAC	Yes	Yes
RAND_bytes	PRNG	Yes	Yes
RAND_seed	PRNG	Yes	Yes
SHA1_Final	SHA1	Yes	Yes
SHA1_Init	SHA1	Yes	Yes
SHA1_Update	SHA1	Yes	Yes

Storing and Loading Cryptographic Information

Serialization

Function Name	Algo	Fmt	Save/Load	Usr	Crp Ofc
DHparams_print	DH	TEXT	SAVE	Yes	Yes
DSA_print	DSA	TEXT	SAVE	Yes	Yes
DSAparams_print	DSA	TEXT	SAVE	Yes	Yes
PEM_write_bio_DHparams	DH	PEM	SAVE	Yes	Yes
PEM_write_bio_DSAPrivateKey	DSA	PEM	SAVE	Yes	Yes
PEM_write_bio_DSA_PUBKEY	DSA	PEM	SAVE	Yes	Yes
PEM_write_bio_DSAparams	DSA	PEM	SAVE	Yes	Yes
i2d_DSAPrivateKey_bio	DSA	DER	SAVE	Yes	Yes
i2d_DSA_PUBKEY_bio	DSA	DER	SAVE	Yes	Yes

Deserialization

Function Name	Algo	Fmt	Save/Load	Usr	Crp Ofc
PEM_read_bio_DHparams	DH	PEM	LOAD	Yes	Yes
PEM_read_bio_DSAPrivateKey	DSA	PEM	LOAD	Yes	Yes
PEM_read_bio_DSA_PUBKEY	DSA	PEM	LOAD	Yes	Yes
PEM_read_bio_DSAparams	DSA	PEM	LOAD	Yes	Yes
d2i_DSAPrivateKey_bio	DSA	DER	LOAD	Yes	Yes
d2i_DSA_PUBKEY_bio	DSA	DER	LOAD	Yes	Yes

Basic Support Functions

Mathematics Data Structures

Function Name	Usr	Crp Ofc
ASN1_INTEGER_free	Yes	Yes
ASN1_INTEGER_get	Yes	Yes
ASN1_INTEGER_new	Yes	Yes
ASN1_INTEGER_set	Yes	Yes
ASN1_INTEGER_to_BN	Yes	Yes
ASN1_TIME_free	Yes	Yes
ASN1_TIME_new	Yes	Yes
ASN1_TIME_set	Yes	Yes

ASN1_UTCTIME_print	Yes	Yes
BN_bin2bn	Yes	Yes
BN_bn2bin	Yes	Yes
BN_bn2dec	Yes	Yes
BN_bn2hex	Yes	Yes
BN_clear	Yes	Yes
BN_clear_free	Yes	Yes
BN_cmp	Yes	Yes
BN_copy	Yes	Yes
BN_dec2bn	Yes	Yes
BN_dup	Yes	Yes
BN_free	Yes	Yes
BN_hex2bn	Yes	Yes
BN_init	Yes	Yes
BN_new	Yes	Yes
BN_num_bits	Yes	Yes
BN_print	Yes	Yes
BN_swap	Yes	Yes
BN_to_ASN1_INTEGER	Yes	Yes
BN_value_one	Yes	Yes
X509_gmtime_adj	Yes	Yes
X509_time_adj	Yes	Yes

Basic Input and Output Operations

Function Name	Usr	Crp Ofc
BIO_free	Yes	Yes
BIO_gets	Yes	Yes
BIO_new_fd	Yes	Yes
BIO_new_file	Yes	Yes
BIO_new_fp	Yes	Yes
BIO_new_mem_buf	Yes	Yes
BIO_puts	Yes	Yes
BIO_read	Yes	Yes
BIO_write	Yes	Yes

Library Information Functions

Function Name	Usr	Crp Ofc
ERR_error_string	Yes	Yes
ERR_error_string_n	Yes	Yes
ERR_free_strings	Yes	Yes
ERR_func_error_string	Yes	Yes
ERR_get_error	Yes	Yes
ERR_lib_error_string	Yes	Yes
ERR_load_crypto_strings	Yes	Yes

ERR_print_errors	Yes	Yes
ERR_reason_error_string	Yes	Yes
getCocoCryptoVersion	Yes	Yes

Advanced Functions

The functions listed under this heading may not be used in FIPS mode.

Advanced Input and Output Operations

Function Name	Usr	Crp Ofc
BIO_callback_ctrl	Yes	Yes
BIO_copy_next_retry	Yes	Yes
BIO_ctrl	Yes	Yes
BIO_ctrl_get_read_request	Yes	Yes
BIO_ctrl_get_write_guarantee	Yes	Yes
BIO_ctrl_pending	Yes	Yes
BIO_ctrl_reset_read_request	Yes	Yes
BIO_ctrl_wpending	Yes	Yes
BIO_debug_callback	Yes	Yes
BIO_dump	Yes	Yes
BIO_dump_indent	Yes	Yes
BIO_dup_chain	Yes	Yes
BIO_f_base64	Yes	Yes
BIO_f_buffer	Yes	Yes
BIO_f_cipher	Yes	Yes
BIO_f_md	Yes	Yes
BIO_f_null	Yes	Yes
BIO_find_type	Yes	Yes
BIO_free_all	Yes	Yes
BIO_get_retry_BIO	Yes	Yes
BIO_get_retry_reason	Yes	Yes
BIO_int_ctrl	Yes	Yes
BIO_new	Yes	Yes
BIO_next	Yes	Yes
BIO_pop	Yes	Yes
BIO_ptr_ctrl	Yes	Yes
BIO_push	Yes	Yes
BIO_s_bio	Yes	Yes
BIO_s_fd	Yes	Yes
BIO_s_file	Yes	Yes
BIO_s_mem	Yes	Yes
BIO_s_null	Yes	Yes
BIO_set	Yes	Yes
BIO_set_cipher	Yes	Yes

Mathematical Operations

Function Name	Usr	Crp Ofc
BN_CTX_free	Yes	Yes
BN_CTX_init	Yes	Yes
BN_CTX_new	Yes	Yes
BN_add	Yes	Yes
BN_add_word	Yes	Yes
BN_clear_bit	Yes	Yes
BN_div	Yes	Yes
BN_div_word	Yes	Yes
BN_generate_prime	Yes	Yes
BN_get_word	Yes	Yes
BN_is_bit_set	Yes	Yes
BN_is_prime	Yes	Yes
BN_lshift	Yes	Yes
BN_mask_bits	Yes	Yes
BN_mod_word	Yes	Yes
BN_mul	Yes	Yes
BN_mul_word	Yes	Yes
BN_rshift	Yes	Yes
BN_set_bit	Yes	Yes
BN_set_word	Yes	Yes
BN_sqr	Yes	Yes
BN_sub	Yes	Yes
BN_sub_word	Yes	Yes

Memory and Data Structure Management Functions

Function Name	Usr	Crp Ofc
CRYPTO_free	Yes	Yes
CRYPTO_malloc	Yes	Yes
sk_delete	Yes	Yes
sk_delete_ptr	Yes	Yes
sk_dup	Yes	Yes
sk_find	Yes	Yes
sk_free	Yes	Yes
sk_insert	Yes	Yes
sk_is_sorted	Yes	Yes
sk_new	Yes	Yes
sk_new_null	Yes	Yes
sk_num	Yes	Yes
sk_pop	Yes	Yes
sk_pop_free	Yes	Yes
sk_push	Yes	Yes

sk_set	Yes	Yes
sk_set_cmp_func	Yes	Yes
sk_shift	Yes	Yes
sk_sort	Yes	Yes
sk_unshift	Yes	Yes
sk_value	Yes	Yes
sk_zero	Yes	Yes

Abstraction Functions

The functions listed under this heading may not be used in FIPS mode.

EVP Functions

EVP Digest Functions

EVP Digest Algorithm Specifiers

Function Name	Usr	Crp Ofc
EVP_dss1	Yes	Yes
EVP_get_digestbyname	Yes	Yes
EVP_sha1	Yes	Yes

EVP Digest Context Management

Function Name	Usr	Crp Ofc
EVP_MD_CTX_cleanup	Yes	Yes
EVP_MD_CTX_create	Yes	Yes
EVP_MD_CTX_destroy	Yes	Yes
EVP_MD_CTX_init	Yes	Yes

EVP Digest Operations

Function Name	Usr	Crp Ofc
EVP_Digest	Yes	Yes
EVP_DigestFinal_ex	Yes	Yes
EVP_DigestInit_ex	Yes	Yes
EVP_DigestUpdate	Yes	Yes
EVP_SignFinal	Yes	Yes
EVP_VerifyFinal	Yes	Yes

EVP Cipher Functions

EVP Key Handling Functions

Function Name	Usr	Crp Ofc
EVP_PKEY_assign	Yes	Yes
EVP_PKEY_bits	Yes	Yes

EVP_PKEY_cmp_parameters	Yes	Yes
EVP_PKEY_copy_parameters	Yes	Yes
EVP_PKEY_free	Yes	Yes
EVP_PKEY_get1_DH	Yes	Yes
EVP_PKEY_get1_DSA	Yes	Yes
EVP_PKEY_missing_parameters	Yes	Yes
EVP_PKEY_new	Yes	Yes
EVP_PKEY_save_parameters	Yes	Yes
EVP_PKEY_set1_DH	Yes	Yes
EVP_PKEY_set1_DSA	Yes	Yes
EVP_PKEY_size	Yes	Yes
EVP_PKEY_type	Yes	Yes
PEM_read_bio_PUBKEY	Yes	Yes
PEM_read_bio_PrivateKey	Yes	Yes
PEM_write_bio_PUBKEY	Yes	Yes
PEM_write_bio_PrivateKey	Yes	Yes

EVP Cipher Algorithm Specifiers

Function Name	Usr	Crp Ofc
EVP_CIPHER_type	Yes	Yes
EVP_aes_256_cfb128	Yes	Yes
EVP_get_cipherbyname	Yes	Yes

EVP Cipher Context Management

Function Name	Usr	Crp Ofc
EVP_CIPHER_CTX_cleanup	Yes	Yes
EVP_CIPHER_CTX_init	Yes	Yes
EVP_CIPHER_CTX_set_key_length	Yes	Yes
EVP_CIPHER_CTX_set_padding	Yes	Yes

EVP Cipher Context Operations

Function Name	Usr	Crp Ofc
EVP_CipherFinal_ex	Yes	Yes
EVP_CipherInit_ex	Yes	Yes
EVP_CipherUpdate	Yes	Yes
EVP_DecryptFinal_ex	Yes	Yes
EVP_DecryptInit_ex	Yes	Yes
EVP_DecryptUpdate	Yes	Yes
EVP_EncryptFinal_ex	Yes	Yes
EVP_EncryptInit_ex	Yes	Yes
EVP_EncryptUpdate	Yes	Yes
EVP_OpenFinal	Yes	Yes
EVP_OpenInit	Yes	Yes
EVP_OpenUpdate	Yes	Yes

EVP_SealFinal	Yes	Yes
EVP_SealInit	Yes	Yes
EVP_SealUpdate	Yes	Yes

EVP Subsystem Support

Function Name	Usr	Crp Ofc
ENGINE_by_id	Yes	Yes
ENGINE_cleanup	Yes	Yes
ENGINE_finish	Yes	Yes
ENGINE_free	Yes	Yes
ENGINE_init	Yes	Yes
ENGINE_load_builtin_engines	Yes	Yes
ENGINE_register_all_ciphers	Yes	Yes
ENGINE_register_all_digests	Yes	Yes
OPENSSL_add_all_algorithms_conf	Yes	Yes
OPENSSL_add_all_algorithms_noconf	Yes	Yes

X.509 Certificate Functions

X.509 Data Structure Creation and Initialization

Function Name	Usr	Crp Ofc
NCONF_WIN32	Yes	Yes
NCONF_default	Yes	Yes
NCONF_new	Yes	Yes
X509V3_EXT_add_nconf	Yes	Yes
X509V3_set_ctx	Yes	Yes
X509V3_set_nconf	Yes	Yes
X509_EXTENSION_dup	Yes	Yes
X509_EXTENSION_new	Yes	Yes
X509_NAME_add_entry_by_NID	Yes	Yes
X509_NAME_add_entry_by_txt	Yes	Yes
X509_NAME_delete_entry	Yes	Yes
X509_NAME_dup	Yes	Yes
X509_NAME_new	Yes	Yes
X509_REQ_new	Yes	Yes
X509_REQ_set_pubkey	Yes	Yes
X509_REQ_set_subject_name	Yes	Yes
X509_REQ_set_version	Yes	Yes
X509_STORE_CTX_init	Yes	Yes
X509_STORE_CTX_new	Yes	Yes
X509_STORE_CTX_set_time	Yes	Yes
X509_STORE_add_cert	Yes	Yes
X509_STORE_new	Yes	Yes
X509_add1_ext_i2d	Yes	Yes

X509_add_ext	Yes	Yes
X509_dup	Yes	Yes
X509_new	Yes	Yes
X509_set_issuer_name	Yes	Yes
X509_set_notAfter	Yes	Yes
X509_set_notBefore	Yes	Yes
X509_set_pubkey	Yes	Yes
X509_set_serialNumber	Yes	Yes
X509_set_subject_name	Yes	Yes
X509_set_version	Yes	Yes

X.509 Data Structure Inspection

Function Name	Usr	Crp Ofc
NCONF_get_number_e	Yes	Yes
NCONF_get_section	Yes	Yes
NCONF_get_string	Yes	Yes
OBJ_ln2nid	Yes	Yes
OBJ_nid2ln	Yes	Yes
OBJ_nid2obj	Yes	Yes
OBJ_nid2sn	Yes	Yes
OBJ_obj2nid	Yes	Yes
OBJ_sn2nid	Yes	Yes
OBJ_txt2nid	Yes	Yes
OBJ_txt2obj	Yes	Yes
X509V3_EXT_get_nid	Yes	Yes
X509_NAME_cmp	Yes	Yes
X509_NAME_entry_count	Yes	Yes
X509_NAME_get_entry	Yes	Yes
X509_NAME_get_index_by_NID	Yes	Yes
X509_NAME_get_text_by_NID	Yes	Yes
X509_REQ_get_pubkey	Yes	Yes
X509_cmp	Yes	Yes
X509_find_by_issuer_and_serial	Yes	Yes
X509_find_by_subject	Yes	Yes
X509_get_issuer_name	Yes	Yes
X509_get_pubkey	Yes	Yes
X509_get_serialNumber	Yes	Yes
X509_get_subject_name	Yes	Yes
X509_verify_cert_error_string	Yes	Yes

X.509 Data Structure Destruction

Function Name	Usr	Crp Ofc
NCONF_free	Yes	Yes
NCONF_free_data	Yes	Yes
X509_EXTENSION_free	Yes	Yes

X509_INFO_free	Yes	Yes
X509_NAME_free	Yes	Yes
X509_REQ_free	Yes	Yes
X509_STORE_CTX_free	Yes	Yes
X509_STORE_free	Yes	Yes
X509_free	Yes	Yes

X.509 Serialization

Function Name	Usr	Crp Ofc
NCONF_dump_bio	Yes	Yes
PEM_X509_INFO_write_bio	Yes	Yes
PEM_write_bio_X509	Yes	Yes
PEM_write_bio_X509_AUX	Yes	Yes
PEM_write_bio_X509_CRL	Yes	Yes
PEM_write_bio_X509_REQ	Yes	Yes
PEM_write_bio_X509_REQ_NEW	Yes	Yes
X509V3_EXT_print	Yes	Yes
X509_NAME_oneline	Yes	Yes
X509_NAME_print	Yes	Yes
X509_NAME_print_ex	Yes	Yes
X509_REQ_print	Yes	Yes
X509_REQ_print_ex	Yes	Yes
X509_print	Yes	Yes
X509_print_ex	Yes	Yes
i2d_X509_REQ_bio	Yes	Yes
i2d_X509_bio	Yes	Yes

X.509 Deserialization

Function Name	Usr	Crp Ofc
NCONF_load	Yes	Yes
NCONF_load_bio	Yes	Yes
PEM_X509_INFO_read_bio	Yes	Yes
PEM_read_bio_X509	Yes	Yes
PEM_read_bio_X509_AUX	Yes	Yes
PEM_read_bio_X509_CRL	Yes	Yes
PEM_read_bio_X509_REQ	Yes	Yes
d2i_X509_REQ_bio	Yes	Yes
d2i_X509_bio	Yes	Yes

X.509 Cryptographic Operations

Function Name	Usr	Crp Ofc
X509_NAME_digest	Yes	Yes
X509_REQ_digest	Yes	Yes
X509_REQ_sign	Yes	Yes

X509_REQ_to_X509	Yes	Yes
X509_REQ_verify	Yes	Yes
X509_check_private_key	Yes	Yes
X509_check_trust	Yes	Yes
X509_digest	Yes	Yes
X509_pubkey_digest	Yes	Yes
X509_sign	Yes	Yes
X509_to_X509_REQ	Yes	Yes
X509_verify	Yes	Yes
X509_verify_cert	Yes	Yes