IBM CryptoLite for C
Version 4.5

(Non-proprietary)

Security Policy*

IBM Crypto Competence Center
Copenhagen

In collaboration with
IBM Research
Zürich Research Lab

December 03, 2007

http://www.ibm.com/security/products/cryptotools.shtml

---

* Policy revision 117

Contents

1. Scope of Document

This document describes the services that the *IBM CryptoLite for C* library ("CLiC", or just "module") provides to security officers and end users, and the policy governing access to those services.

Descriptions in this policy are specifically applicable to the module version being validated (4.5). Other versions of the module have been released, including a previously FIPS-validated one; where applicable, references are made to differences.
There is a single, non-proprietary version of the security policy (i.e., this document).

Module Description The IBM CryptoLite for C library in its FIPS configuration consists of a single loadable module, a shared library.

This validation targeted the following builds: x86/Linux ("Lintel") (tested by laboratory), Windows Vista (tested by laboratory), AIX (vendor affirmed).

The CLiC API represents the logical boundary of the module. The physical cryptographic boundary for the module is defined as the enclosure of the host on which the cryptographic module is to be executed.

2. Cryptographic Module Specification

The IBM CryptoLite for C module is classified as a multi-chip standalone module for **FIPS Pub 140–2** purposes. As such, the module must be validated upon particular operating systems and computer platforms. The actual cryptographic boundary for this FIPS validation thus includes the CLiC module running in the following configurations:

1. IBM-compatible PC running Red Hat Enterprise Linux ES release 4 (Nahant Update 4) x86
2. IBM-compatible PC running Windows Vista Ultimate x86

The exact module configuration is implicitly described by the cryptographic hashes of the validated configuration:

1. Lintel DLL:
   `SHA-1 hash 89be10472 ae53fbdb7 18f40f154 ebac1753d 9145,`
   `SHA-256 hash eb9782a0b 575d4ec4e 61777a39d b2f9fd686 1fded7731 36f53b6a3 319a34271e.`
2. Windows DLL:
   `SHA-1 hash 17608b584 fe03e5c22 048d577de b752faaad 70ec,`
   `SHA-256 hash 75c3b503d baf3a7b22 38103da36 4200c1583 298bb930d 2fba6c892 dffc872cfd.`
3. AIX (vendor affirmed) object file:
   `SHA-1 hash 205b6ec73 e25b66d3f 2930b988c b124edba3 f451,`
   `SHA-256 hash 5e02cb791 dfd26ad83 f836db9e9 486c25c09 e9fd4a190 2b8dafc59 c74ea33114.`

The module running on the above platforms was validated as meeting all FIPS **Pub 140–2** Level 1 security requirements. The CLiC module is packaged in a single DLL which contains all the code for the module. The library is accompanied by its primary header file, `clic.h`. (Other support files, such as auxiliary headers or link files, may also be included in the distribution.) Actual DLL name is system-specific; some of the possible names are `libclic.a, libclic.so` (possibly as `libclic.so.NNN`), or `clic.dll`.

In addition to configurations tested by the laboratory, IBM tested library instances on the following platforms (*all vendor affirmed*):

1. IBM-compatible PC with Red Hat Enterprise Linux ES release 4 (Nahant Update 4) x86-64
2. IBM System p running Red Hat Enterprise Linux ES release 4 (Nahant Update 4) PPC32
3. IBM System p running Red Hat Enterprise Linux ES release 4 (Nahant Update 4) PPC64
4. IBM System p running AIX 5L 5.2 PPC32
5. IBM System p running AIX 5L 5.2 PPC64
6. IBM System z running SUSE Linux Enterprise Server (SLES) 9 s390x

IBM vendor affirms that binaries on these platforms operate correctly, and thus maintains their FIPS compliance.

The CLiC library also runs on many other platforms, including other AIX versions, MVS and USS32 (on mainframes), Solaris, other Windows variants, HP-UX, and PalmOS. Validation testing did not cover these platforms.

Security level This document describes the security policy for the IBM CryptoLite for C with Level 1 overall security as defined in **FIPS Pub 140-2**[5].

Module components

Type	Name	Release	Date	SHA-256 hash
Linux versions				
Software (DLL)	`libclic.so.4`	4.5	2007.04.01	see above
Documentation	`CLiC User Guide`	4.5	2007.04.01	N/A
Windows				
Software (DLL)	`clic.dll`	4.5	2007.04.01	see above
Documentation	`CLiC User Guide`	4.5	2007.04.01	N/A
AIX configurations (vendor affirmed)				
Software (DLL)	`libclic.so`	4.5	2007.04.01	see above
Documentation	`CLiC User Guide`	4.5	2007.04.01	N/A

3. Cryptographic Module Security Level

The module is intended to meet requirements of Security Level 1 overall, with certain categories of security requirements not applicable (Table 1).

Security Requirements Section	Level
Cryptographic Module Specification	3
Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	(1)
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A

Table 1: Module Security Level Specification.

EMI/EMC properties of the IBM CryptoLite for C are not meaningful for the library itself. System utilizing CLiC library services have their overall EMI/EMC ratings determined by the host system. Validation environments have FCC Class A ratings.

Physical security parameters are inherited from the host system. The library itself has no physical security characteristics.

4. Ports and Interfaces

As a multi-chip standalone module, the *CLiC physical interfaces* are the boundaries of the host running CLiC library code. The underlying *logical interface of the module is C language Application Program Interface (API)*, documented in the CLiC Library Reference Manual.

Control inputs are provided through dedicated functions of the public API. Generally, for most security functions, a setup function performs initialization tasks (key import, key expansion, object initialization, etc.). Such control functions provide no cryptographic services themselves, but they are prerequisites of cryptographic operations.

Data input and *data output* are provided in the variables passed with API calls, generally through user-supplied buffers. The module does not manage memory itself; all input and output is constrained in user-supplied data regions. Special-purpose code tracks that the library operations do not influence memory beyond the limitations described by the user.

Status output is provided in return values documented for each call. Dedicated diagnostics functions generally return more detailed information than cryptographic functions, which primarily indicate success or type of failure.

The module is accessed from C/C++-language programs using the same method as the CLiC static toolkit, via the inclusion of the include file `clic.h`. A companion header, `clic.hpp`, provides a wrapper for pure C++ compilers.

Module Status The CLiC communicates any error status asynchronously through the use of its documented return codes. It is the responsibility of the calling application to handle exceptions in a FIPS 140 appropriate manner.

In addition to failures producing error codes, the module is equipped with internal consistency checks (“assertions”) along its control path, monitoring the consistency of module internals. Failure of internal checks is reported as an unexpected error condition, and terminates the CLiC instance. These exceptions provide system-level failure notification, not just CLiC errors.

5. Roles, Services, and Authentication

5.1 Roles

The module supports two roles, a *cryptographic officer role* and a *user role* (Table 2). Roles are not explicitly authenticated; the capability to invoke the corresponding instructions implicitly authenticates users (i.e., callers).

The *officer role* is a purely an administrative role that does not involve the use of cryptographic services. The role is not explicitly authenticated but assumed implicitly on implementation of the modules installation and usage sections defined in the security rules section.

The *user role* has access to all of the modules services. The role is not explicitly authenticated but assumed implicitly on access of any of the non-officer services.

Role	Type of Authentication	Authentication Data	Strength of mechanism
Officer	None (automatic)	None	N/A
User	None (automatic)	None	N/A

Table 2: Roles and Authentication mechanisms.

5.2 Services

The module provides *queries* and *commands* (Table 6 and 5). Queries return status of commands or command groups; commands exercise cryptographic functions. The officer performs queries; users may access both queries and commands. Certain test queries are executed automatically or usually not as part of regular operations; these special cases are parenthesized as “(yes)” in Table 6.

Module services are accessed through documented API interfaces from the calling application.

All algorithms support all combinations of key sizes and modes, where applicable. Algorithms labelled *“legacy”* are supported only for backwards compatibility with existing applications. Elliptic curve (EC) support is limited to prime fields (FP), using curves in Table 7. The supported NIST P-curves have corresponding SECG or ANSI equivalents [1, 2, 4].

In addition to low-level primitives (such as “AES in elementary modes (ECB/CBC)”), the library offers abstract access to encryption and digest functions. The `CLiC cipher NNN` functions provide transparent access to encryption, specifying type indirectly (i.e., creating a cipher object of type `CLiC T AES`). (A similar abstraction is possible for digesting through `CLiC digest NNN` functions.)

Functions in generic cipher and digest operations observe low-level restrictions indirectly, as they access the corresponding primitives internally. The generic operations themselves enforce certain restrictions that the algorithm primitives do not allow (such as rejecting non-approved modes of operation, where applicable).

Similar to encryption and digesting, public-key operations have similar abstractions. Generic public-key operations (such as functions in the `CLiC pk NNN` group) perform basic operations (encrypt, decrypt, sign, verify) without requiring direct access to low-level primitives (such as `CLiC rsaSign`). These functions restrict non-approved algorithms; the underlying functions provide further restrictions (such as checking padding modes).

Authorized service	Officer	User
Officer services		
Invoke FIPS self-tests	Yes	No
User services		
AES encryption/decryption	No	Yes
TDES encryption/decryption	No	Yes
DES encryption/decryption (single-DES, for compatibility with legacy applications)	No	Yes
CAST-5 (CAST-128) encryption/decryption	No	Yes
CAST-6 (CAST-256) encryption/decryption	No	Yes
RC2 encryption/decryption (legacy algorithm)	No	Yes
“ArcFour” encryption/decryption	No	Yes
Blowfish encryption/decryption	No	Yes
DSA parameter/key generation	No	Yes
DSA signature generation and verification	No	Yes
RSA signature generation and verification	No	Yes
RSA key generation (**ANSI X9.31**)	No	Yes
Elliptic Curve (EC) signature generation and verification (**ECDSA**)	No	Yes
EC key generation	No	Yes
EC Diffie-Hellmann (**ECDH**) key agreement	No	Yes
Diffie-Hellmann (DH) key agreement (incl. parameter generation)	No	Yes
RSA sign/verify (non-approved schemes)	No	Yes
RSA encrypt/decrypt	No	Yes
SHA-1 hash	No	Yes
SHA-224 hash	No	Yes
SHA-256 hash	No	Yes
SHA-384 hash	No	Yes
SHA-512 hash	No	Yes
MD5 hash (legacy algorithm)	No	Yes
Whirlpool hash (ISO and non-ISO)	No	Yes
MD2 hash (legacy algorithm)	No	Yes
HMAC-SHA message authentication (all supported SHA versions)	No	Yes
CMAC message authentication (utilizing AES or TDES)	No	Yes
HMAC (non-SHA) message authentication	No	Yes
DRNG, obtain random number (FIPS 186-2/ANSI X9.31 generator)	No	Yes
TRNG, generate random seed (internal TRNG)	No	Yes
Generic functional calls (encrypt, digest, public-key, PKI)	No	Yes
Format conversions (non-cryptographic)	No	Yes
Other auxiliary functions	No	Yes

Table 3: Services by role

Service	Notes	Modes	Approved?	Role	
				Officer	User
Symmetric encryption and decryption					
AES	128, 192, or 256 bit keys (FIPS 197)	**ECB, CBC, CCM, CTR**	**yes**	no	yes
	128, 192, or 256 bit keys	GCM, CTS	no	no	yes
TDES	112 or 168 bit keys	**ECB, CBC, CTR**	**yes**	no	yes
DES (single-DES)	56 bit keys (legacy)	ECB, CBC, CTR	no	no	yes
CAST-5 (CAST-128)	40 to 128 bit keys	ECB, CBC	no	no	yes
CAST-6 (CAST-256)	128 to 256 bit keys	ECB, CBC	no	no	yes
RC2	up to 128 bit keys	ECB, CBC	no	no	yes
"ArcFour"	up to 256 bit keys	N/A (stream)	no	no	yes
Blowfish	N/A	N/A	no	no	yes
Public-key algorithms					
DSA key/prm generation	1024 bit modulus (FIPS 186-2 key size)	N/A	**yes**	no	yes
DSA sign/verify	N/A	N/A	**yes**	no	yes
RSA sign/verify	ANSI X9.31, PKCS #1	N/A	**yes**	no	yes
RSA key generation	ANSI X9.31	N/A	**yes**	no	yes
EC key generation	NIST P-192 to P-521	N/A	**yes**	no	yes
ECDSA sign/verify	NIST P-192 to P-521	N/A	**yes**	no	yes
ECDH key agreement	NIST P-192 to P-521	N/A	**yes**	no	yes
Diffie-Hellmann (DH)	1024 or 2048 bits modulus	key agreement	**yes**	no	yes
Diffie-Hellmann (DH)	512 bits (legacy compatibility)	key agreement	no	no	yes
RSA sig/ver (non-appr)	ISO 9796	N/A	no	no	yes
RSA encrypt/decrypt	PKCS 1, OAEP	N/A	no	no	yes
Hash functions					
SHA-1	FIPS 180–1	N/A	**yes**	no	yes
SHA-224	FIPS 180–2 (2004.02 change notice)	N/A	**yes**	no	yes
SHA-256	FIPS 180–2	N/A	**yes**	no	yes
SHA-384	FIPS 180–2	N/A	**yes**	no	yes
SHA-512	FIPS 180–2	N/A	**yes**	no	yes
MD5	RFC 1321 (legacy)	N/A	no	no	yes
Whirlpool	ISO and NESSIE variants	N/A	no	no	yes
MD2	RFC 1319 (legacy)	N/A	no	no	yes
Message Authentication Codes (MACs)					
HMAC-SHA	with SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512	N/A	**yes**	no	yes
CMAC	with AES or TDES	(CMAC)	**yes**	no	yes
HMAC (non-SHA)	with MD5	N/A	no	no	yes
Random number generation					
DRNG	FIPS 186-2, ANSI X9.31	N/A	**yes**	no	yes
TRNG	entropy extractor	N/A	no	no	yes
Other functions					
Generic functional calls	encrypt, digest, PK, PKI	N/A	N/A	no	yes
Format conversions	non-cryptographic conversion	N/A	N/A	no	yes
Other functions	(see documentation)	N/A	N/A	no	yes

Table 5: Commands, grouped by functionality

Service	Notes	Role	
		Officer	User
Module status			
Query mode	Check if module is still in FIPS mode. `(CLiC fips140 status)`	yes	no
Integrity checks			
Power-up test	automatic before first use includes `CLiC fips140 checkIntegrity`	(yes)	no
Self-tests	`CLiC fips140 selftest`	yes	no
Operational correctness checks			
RNG tests	continuously performed (automatic)	N/A	N/A
Comprehensive test application			
test-clic application	generated at CLiC build time very high coverage (*external utility*)	(yes)	no

Table 6: Queries

Curve	Key length (bits)	Strength (bits)	Notes
NIST P-192	192	80	SEC2 secp192r1, ANSI X9.62 prime192v1
NIST P-224	224	112	SEC2 secp224r1
NIST P-256	256	128	SEC2 secp256r1, ANSI X9.62 prime256v1
NIST P-384	384	192	SEC2 secp384r1
NIST P-521	521	256	SEC2 secp521r1

Table 7: EC curve support

In addition to generic public-key operations, a related set of higher-level functions provide generic PKI integration services. PKI services are abstractions over the level of algorithm approval, and only verify algorithm permissions indirectly. PKI services rely on ASN.1/BER format conversions, which are also generally outside algorithm approval.

Single DES is optionally available, labelled as non-approved, for legacy compatibility purposes. A dedicated build option enables CLiC builds to specifically include single-DES, as a non-approved algorithm. The module under validation includes single-DES, flagging it as non-approved.

Format conversions, labelled as “other operations”, are non-cryptographic commands that change the representation of data. Format converters read and write, among others, the following formats:

- Various protocols based on ASN.1/BER encoded data (PKI-related and similar standard formats) Custom BER/DER encodings may be supported through direct access to two direct ASN.1-encoding functions.
- Conversions between industry-standard *object identifiers* and CLiC internal symbolic constants
- Base-64 encoding (“ASCII armor”), generating and reading printable representation of binary data
- Multibyte encoding of text, such as UTF-8 and Unicode subsets

- Conversions between ASCII and non-ASCII data (such as EBCDIC).
  Note: most relevant operations tolerate both ASCII and EBCDIC input. Explicit conversion functions are also available.

Format conversion services do not provide cryptographic functionality, but may use other services, if the transport mechanism requires them. As an example, if *signed data* is represented as a standard ASN.1 structure, it uses one of the *sign* or *verify* calls, implicitly.

A few services, such as those related to internal token manipulation, are also grouped under “other functions”. All such functions are described in the product documentation, but they are not discussed in this document.

The module does not explicitly identify or authenticate users for any of the roles.

6. Operational Environment

The CLiC security module is written mostly in C, and has been extensively reviewed to confirm security. Extensive internal consistency checks verify both user input and library configuration, terminating early if errors are encountered. Buffer handling, one of the most problematic parts of C development, is kept out of CLiC scope, since all persistent storage is managed by callers.

The module implements both approved and non-approved services. The calling application controls the cryptographic material as well as the services that use them. It is the applications responsibility to ensure that when in a FIPS compliant mode, only those approved algorithms are used. The FIPS configuration automatically inhibits parameter combinations that are technically possible but outside the standardized range (such as DSA keys over 1024 bits, very short HMAC keys, etc.). If non-approved algorithms are used, the module will switch to non-FIPS mode and stay there until re-initialized.

CLiC is developed and maintained according to IBM internal development standards. Industry-standard tools, including CVS (Version 1.11.21 as of this writing) are used for configuration management. Version control covers source code, test data, and support documentation.

6.1 Key Management

Key Storage The CLiC library does not provide internal long-term cryptographic key storage; all persistent storage is managed by applications. It is the responsibility of the application program developers to ensure **FIPS Pub 140–2** compliance of key storing techniques they implement.

The module provides applications key import and export routines such that key material can be used in conjunction with cryptographic services. *It is the responsibility of applications using library services to ensure that these services are used in a FIPS compliant manner.* Keys so managed or generated by applications or libraries may be passed from application to the module in the clear, provided that the sending application or library exists within the physical boundary of the host computer.

Key Generation Key Generation uses an approved RNG (specified both in **FIPS Pub 186–2** and ANSI X9.31) algorithm which is based on SHA-1. The DRNG has a maximum number of internal states of 2^{160}, this being limited by the compression function in SHA-1. RSA and DH key generation algorithms use the DRNG engine seeded with 20 bytes of true random data. This true random generator is based on IBM patented technology where statistical analysis used to estimate the entropy of clock jitter. The internal TRNG engine defaults to an automatic reseeding policy that adds a true random byte every 128 bytes of output, or if a given number of seconds has passed since the last seeding. Applications can additionally provide their own seeding data and also increase the automatic reseeding frequency of the internal RNG.

DSA key generation is compliant with **FIPS Pub 186–2**. *In FIPS mode, RSA key generation only implements the ANSI X9.31 key generation method [3].* A non-compliant RSA key generation method may also be present in non-FIPS versions of CLiC, for RSA keys shorter than 1024 bits (ANSI X9.31 does not permit generation of shorter keys), but the FIPS configuration does not permit the generation of such short keys. (This non-approved key generation method, superseded by the ANSI X9.31 algorithm, is retained for compatibility with previous releases, but it is not generally available in recent builds.)

Key Establishment *Using Diffie-Hellmann (DH) key establishment, predefined DH constants are available, 1024 or 2048 bit moduluses in approved mode (80 and 112 bits strength, respectively)*. The legacy, non-approved 512 bit mode of DH key establishment provides 56 bits of encryption strength.

Using ECDH, standard NIST P-curves are used (P-192 to P-521, see Table 7). *ECDH key establishment provides strength from 80 to 256 bits.*

RSA-based key establishment has no fixed upper limit on modulus size. The approved mode provides 80-bits of security strength and since it has no upper bound it can protect a symmetric key up to 256-bits, which provides coverage for all symmetric key sizes. The non-approved security mode provides less than 80-bits of security when RSA key sizes less than 1024-bits are used.

Key Protection *To enforce compliance with FIPS 140–2 key management requirements on the CLiC library itself, code issuing CLiC calls must manage keys in a FIPS 140–2-compliant method. Keys so managed or generated by applications may be passed from the application to the in the clear in the FIPS validated configuration.*

The management and allocation of memory is the responsibility of the operating system. It is assumed that a unique process space is allocated for each request, and that the operating system and the underlying hardware control access to that space. Each instance of the cryptographic module is self-contained within a process space; the library relies on such process separation to maintain confidentiality of secrets. *All platforms used during FIPS validation provide per-process protection for user data.*

All keys are associated with the User role. It is the responsibility of application program developers to protect keys exported from the CLiC module.

Key Destruction Applications must destroy persistent key objects and similar sensitive information through **FIPS Pub 140–2** compliant procedures. The CLiC library itself does not destroy keys and secrets, as it does not own or discard persistent objects. Objects, when released on behalf of a caller, are wiped before they are released.

6.2 Physical Security

The CLiC installation inherits the physical characteristics of the host running it.

6.3 EMI/EMC

EMI/EMC properties of the CLiC deployment are identical to those of the host server or client.

7. Self-tests

The CLiC library implements a number of self-tests to check the proper functioning of the module. This includes power-up self-tests and conditional self-tests. Conditional tests are performed when symmetric or asymmetric keys are generated. These tests include a continuous random number generator test (see details below) and pair-wise consistency tests of the generated DSA or RSA keys.

Startup Self-Tests "Power-up" self-tests are performed automatically when the CLiC library starts loading. (See the Finite State Machine for more details). These tests comprise of the software integrity test and the known answer tests of cryptographic algorithms. Should any of these tests fail; the CLiC module will terminate the loading process. The module cannot be used in this state.

The integrity of the module is verified by checking a SHA-256-based HMAC of the module binary. This 256 bits HMAC SHA-256 key is included in the module binary (DLL) in clear text and is loaded with the module. The HMAC field is prepared during shared library (DLL) generation, during the last step of building. Initialization will only succeed if this HMAC is valid. (Integrity verification is contained in `CLiC_fips140_checkIntegrity.`) The Crypto-Officer can remove this key by uninstalling the module from the host computer

The module tests the following cryptographic algorithms: AES, TDES, DES, DSA (sign/verify), ECDSA (sign/verify), RSA (sign/verify, encrypt/decrypt), ECDH (key agreement), SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, CMAC (AES or TDES), HMAC (all SHA-based variants), and the DRNG.

Self tests are performed in logical order, verifying library integrity incrementally:

1 Known-answer test on SHA-256.

2 Known-answer test on HMAC/SHA-256.

3 Integrity test on library, using HMAC/SHA-256.

4 Known-answer tests on remaining algorithms, from integrity-verified binary.

Startup recovery Should the startup self tests fail during module initialization the crypto officer should re-initialize the complete application. The library will reject calls in this state, since it will verify that self-tests have passed before performing cryptographic functions.

Conditional Self-Testing This includes *continuous DRNG testing*. The DRNG generates output in 160-bit blocks, as it is based on a SHA-1 core. Each newly generated block is compared to the previous one. If the DRNG outputs identical blocks, the Module enters the "Conditional Error" state, ceases cryptographic processing, inhibits all data output, and returns an error indicator. It is the responsibility of the calling application to handle the exception in a FIPS-140 appropriate manner, for example by reinitializing the RNG object.

Similar to the DRNG, high-entropy seed extracted by the TRNG is checked for repeated blocks, before seeding the DRNG. If 160-bit blocks of entropy repeat, the TRNG reports a failure, which caller applications must also handle as an exception.

Pair-wise Consistency Checks The test is run whenever the module generates a private key. The private key structure of the module always contains either the data of the corresponding public key or information sufficient for computing the corresponding public key.

Invoking FIPS self-tests on demand If a user can access CLiC services, the library must have passed its HMAC-based integrity check at startup (a prerequisite of successful loading). During regular operations, once the library has become operational, one can always invoke the `CLiC fips140 selftest` function to repeat the required KATs on demand. If these checks pass, the module is working properly.

8. Operational recommendations (Officer/User guidance)

8.1 Module Configuration for FIPS Pub 140–2 Compliance

To verify FIPS-compliant usage, the following requirements must be observed:

- Administrators and users of CLiC should verify the SHA-256 hash of the executable image. If the hash matches the expected value, and the module passes its startup integrity tests, one must verify that the module is still in FIPS mode (through the `CLiC fips140 status query`).

  Note that the HMAC-based integrity check effectively tests the integrity of a different SHA-256 hash over the entire binary. Checking the hash of the library itself verifies that the validated configuration of CLiC is present. (For legacy systems without a `sha256sum` utility, a SHA-1 hash is provided for `sha1sum use`.)

- Applications and libraries using CLiC features must observe FIPS rules for key management and provide their own self-tests.
  For proper operations, one must verify that applications comply with this requirement. While details of these application requirements are outside the scope of this policy, they are mentioned here for completeness.

- The operating system hosting the CLiC library must be set up in accordance with **FIPS Pub 140–2** rules. It must provide sufficient separation between processes to prevent inadvertent access to data of different processes. (This requirement is met for all platforms tested during validation.)

  The module must not be used by multiple callers simultaneously such that they may interfere with each other. Note that since CLiC operates entirely in caller-provided storage, this requirement is automatically met if the OS provides sufficient process separation (since each memory region's, i.e., object ownership is uniquely determined).

- Applications using CLiC services must verify that ownership of keys is not compromised, and keys are not shared between different users of the calling application.

  Note that this requirement may not be enforced by the CLiC library itself, just the application providing the keys to CLiC. It is noted here for the sake of completeness.

- Applications utilizing CLiC services must avoid using non-approved algorithms or modes of operation, if possible. If this is not feasible, the applications must indicate that they utilize non-approved cryptographic services.

- To be in FIPS mode, the CLiC installation must run on a host with commercial grade components, and must be physically protected as prudent in an enterprise environment.

8.2 Determining Mode of Operation

The module provides a dedicated status query in its FIPS builds, the `CLiC fips140 status` function. This function will start indicating FIPS mode after all selftests are successfully completed. *After performing the first operation with a non-approved algorithm or mode of operation, the module leaves FIPS mode, and stays so*. To get the module back to FIPS mode, one must re-instantiate the library (i.e., reload it). Note that importing keys or initializing objects of non-approved services does not switch to non-FIPS mode; only actual data operations do.

Applications utilizing CLiC services must enforce key management compliant with FIPS 140–2 requirements. This should be indicated in an application-specific way that is directly observable by administrators and end-users.

While such application-specific details are outside the scope of the CLiC validation, they are mentioned here for completeness.

The CLiC module implements both approved and non-approved services. The calling application controls the cryptographic material as well as the services that use them. It is an application responsibility to ensure that when in a FIPS compliant mode, only those approved algorithms are used. Non-approved modes of operation are also indicated in the global FIPS mode indicator.

While the underlying library routines are highly configurable, the FIPS configuration automatically inhibits parameter combinations that are technically legal but outside standardized range (such as DSA keys over 1024 bits, very short HMAC keys, etc.). Product documentation describes these additional limitations, recommending to stay within FIPS limits, even if they may be violated without losing cryptographic functionality.

8.3 Testing/Physical Security Inspection Recommendations

In addition to automatic tests, described elsewhere in this document, CLiC users may invoke FIPS mode self-tests at any time. This is initiated through a dedicated function (`CLiC fips140 checkIntegrity`), which gets invoked automatically at startup. *Continuous tests* are part of the corresponding functions, are implicitly enabled in FIPS builds, and are otherwise not observable (unless, of course, when a failure is detected).

For maximal test coverage, one may also use the `test-clic` auxiliary application. This extension is customized to each CLiC instance, providing extensive test coverage of the generated library, beyond **FIPS Pub 140–2** requirements. The test application links against the FIPS library, and exercises the same instance which is used by application using the CLiC library. (Note that comprehensive testing requires considerable time and processor resources.)

Apart from prudent security practice of server applications, and those of security-critical embedded systems (such as PDAs), no further restrictions are placed on hosts utilizing CLiC services.

Function listing

Public functions

CLiC_aes
CLiC_aesKey
CLiC_arcfour
CLiC_arcfourKey
CLiC_attr
CLiC_base64_decode
CLiC_base64_encode
CLiC_blob
CLiC_cast5
CLiC_cast5Key
CLiC_cast6
CLiC_cast6Key
CLiC_cert
CLiC_cert_decrypt
CLiC_cert_isSignerOf
CLiC_cert_verify
CLiC_cipher
CLiC_cipher_clearKeyMaterial
CLiC_cipher_encode
CLiC_cipher_final
CLiC_cipher_getComp
CLiC_cipher_getSize
CLiC_cipher_new
CLiC_cipher_reset
CLiC_cipher_setComp
CLiC_cipher_update
CLiC_cmac
CLiC_cmac_clearKeyMaterial
CLiC_cmac_getComp
CLiC_cmac_new
CLiC_cmac_reset
CLiC_cmac_update
CLiC_compare
CLiC_context
CLiC_context_getComp
CLiC_context_new
CLiC_context_objCount
CLiC_context_setMutex
CLiC_copy
CLiC_crl
CLiC_crl_revokes
CLiC_des
CLiC_desKey
CLiC_des_checkKeyMaterial
CLiC_dh
CLiC_dhKeyGen
CLiC_dhParamGen
CLiC_digest
CLiC_digest_getComp
CLiC_digest_new
CLiC_digest_reset
CLiC_digest_typeComp
CLiC_digest_update
CLiC_dispose
CLiC_dsaKeyGen
CLiC_dsaParamGen
CLiC_dsaSign
CLiC_dsaVerify
CLiC_errnoInfo
CLiC_fips140_checkFileIntegrity
CLiC_fips140_checkIntegrity
CLiC_fips140_initFileIntegrity
CLiC_fips140_initIntegrity
CLiC_fips140_policy
CLiC_fips140_selftest
CLiC_fips140_status
CLiC_getRefCount
CLiC_getType
CLiC_hmac
CLiC_hmac_clearKeyMaterial
CLiC_hmac_getComp
CLiC_hmac_new
CLiC_hmac_reset
CLiC_hmac_update
CLiC_krb5_deriveKeyMaterial
CLiC_krb5_deriveRandom
CLiC_krb5_nfold
CLiC_link
CLiC_lock
CLiC_md2
CLiC_md2Init
CLiC_md5
CLiC_md5Init
CLiC_p10
CLiC_p12
CLiC_p12_add
CLiC_p12_encode
CLiC_p12_new
CLiC_p7
CLiC_p7_add
CLiC_p7_detachContent
CLiC_p7_digest
CLiC_p7_encode
CLiC_p7_envelop
CLiC_p7_new
CLiC_p7_sign
CLiC_pbCipher
CLiC_pbCipher_new
CLiC_pbHmac
CLiC_pbHmac_new
CLiC_pb_keyDerivation
CLiC_pk
CLiC_pk_decrypt
CLiC_pk_dh
CLiC_pk_encode
CLiC_pk_encrypt
CLiC_pk_gen
CLiC_pk_getComp
CLiC_pk_getMaterial
CLiC_pk_new
CLiC_pk_setMaterial
CLiC_pk_sign
CLiC_pk_validate
CLiC_pk_verify
CLiC_pkikey
CLiC_pkikey_encode
CLiC_pkikey_gen
CLiC_pkikey_new
CLiC_pkiobj_addAttr
CLiC_pkiobj_fingerprint
CLiC_pkiobj_getComp
CLiC_pkiobj_getNext
CLiC_pkiobj_scanf
CLiC_pkiobj_setComp
CLiC_pkiobj_unlink
CLiC_pkiobj_writef
CLiC_rc2
CLiC_rc2Key
CLiC_rng
CLiC_rng_byte
CLiC_rng_new
CLiC_rng_policy
CLiC_rng_seed
CLiC_rsaDecrypt
CLiC_rsaEncrypt
CLiC_rsaKeyGen
CLiC_rsaSign
CLiC_rsaVerify
CLiC_setMutex
CLiC_sha
CLiC_sha224
CLiC_sha224Init
CLiC_sha256
CLiC_sha256Init
CLiC_sha384
CLiC_sha384Init
CLiC_sha512
CLiC_sha512Init
CLiC_shaInit
CLiC_token
CLiC_token_add
CLiC_token_decrypt
CLiC_token_verify
CLiC_unlock
CLiC_x500name_utf8
CLiC_x500name_writef

Module code is mainly self contained. Apart from compiler-provided services, standard `libc` requirements include basic string-handling functions and standard memory manipulation (`memcpy`, `memset`, and related functionality). In certain instances, depending on host compiler, many of these basic calls may be resolved at compile time, minimizing actual `libc` invocations.

In addition to basic string/buffer manipulation, the module depends on `libc` providing standard `malloc` and `free` calls. Allocated storage belongs to the calling application, but allocation and release calls are issued from within the library as part of object lifecycle management. Dynamic local allocation, possibly including `alloca`, is handled at compile time and does not present a runtime requirement.

Logically, applications find all public symbols (prototypes, interface constants, predefined sizes etc.) within `clic.h`. All function bodies are contained within the library object file; no code is instantiated from the header file.

Module code is mainly OS-neutral, with specific exceptions related to startup activities: code to launch automatic startup tests is OS-dependent. All other functionality is OS-neutral, with details hidden by `libc`.

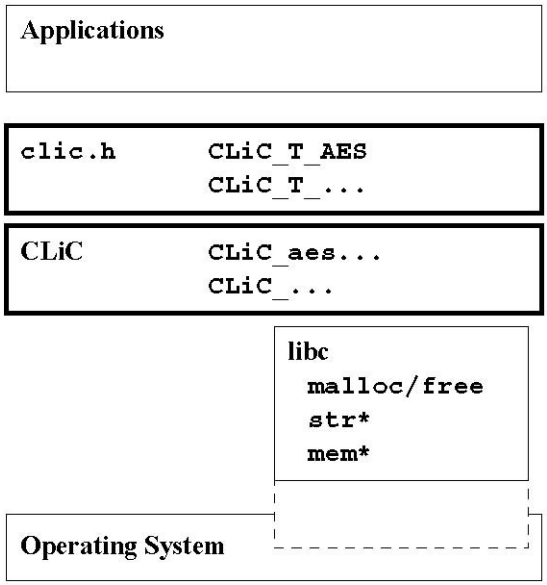

Figure 1: OS/libc dependencies

Glossary

DLL Dynamic Link Library, shared program library instantiated separately from binaries using it. FIPS configurations of CLiC are DLLs, never statically linked.

DRNG Deterministic Random Number Generator, a deterministic function expanding a “true random” seed to a pseudorandom sequence.

FP Prime Field, an elliptic curve (EC) subtype supported by CLiC for key agreement and digital signatures.

KAT Known Answer Test

MVS One of the operating environments used on IBM mainframes. The native MVS version of CLiC was tested by IBM during this FIPS validation process.

OS Operating System

PDA Personal Digital Assistant. Certain ports of CLiC are used on PDAs (for example, Palm or Windows Mobile).

TRNG True Random Number Generator, a service that extracts cryptographically useful random bits from non-deterministic (physical) sources. These “random seed” bits are post-processed by a DRNG.

USS UNIX System Services, a certified UNIX environment running under z/OS on mainframes. USS provides a UNIX interface to native mainframe resources. (USS is independent of z/Linux.)

References

[1] American Bankers Association. *ANSI x9.62, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 1999.

[2] American Bankers Association. *ANSI x9.63, Elliptic Curve Key Agreement and Key Transport Protocols,* 1999.

[3] American National Standards Institute. *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (X9.31),* 1998.

[4] National Institute of Standards and Technology. *Recommended Elliptic Curves for Federal Government Use,* 1999.

[5] National Institute of Standards and Technology. *Security Requirements for Cryptographic Modules* (FIPS 140-2), 2001.