IBM CryptoLite for Java
Version 4.2

(Non-proprietary)

Security Policy *

IBM Crypto Competence Center
Copenhagen

May 26, 2008

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

*Policy revision 35.

Contents

1 Scope of Document **3**

2 Cryptographic Module Specification **3**

3 Cryptographic Module Security Level **6**

4 Ports and Interfaces **7**

5 Roles, Services, and Authentication **8**

5.1 Roles 8

5.2 Services 8

6 Operational Environment **10**

6.1 Key Management 10

6.2 Physical Security 10

6.3 EMI/EMC 11

7 Self-tests **12**

8 Operational recommendations (Officer/User guidance) **13**

8.1 Module Configuration for **FIPS Pub 140–2** Compliance 13

8.2 Determining Mode of Operation 13

8.3 Testing/Physical Security Inspection Recommendations 13

9 Glossary **14**

References **15**

1 Scope of Document

This document describes the services that the *IBM CryptoLite for Java* library ("CLiJ", or just "module") provides to security officers and end users, and the policy governing access to those services. Officers and users are defined in the context of a Java crypto provider (JCE), a cross-platform library providing cryptographic functions [5].

Descriptions in this policy are specifically applicable to the module version being validated (v4.2). Other versions of the module have been released; 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 Java library in its FIPS configuration consists of a single distribution (`jar`) named `c4clij.jar` on systems running Java 5 or above. The binary is cross-platform; the testing laboratory has verified functionality on the following configuration:

1. IBM-compatible PC running Windows Vista Ultimate x86 using Sun JDK 6.0

and remains compliant when running on any system which is binary compatible to the above.

In addition to the above environment, IBM performed testing on the following platforms and vendor-affirms that the binary operates correctly, and thus maintains its FIPS compliance.

1. IBM-compatible PC running Red Hat Enterprise Linux 4 (Nahant Update 4) x86, IBM SDK 5.0, or Sun JDK 5.0, 6.0
2. IBM-compatible PC running Red Hat Enterprise Linux 4 (Nahant Update 4) x86_64, IBM SDK 5.0, or Sun JDK 5.0, 6.0
3. IBM pSeries running Red Hat Enterprise Linux 4 (Nahant Update 4) PPC32 using IBM SDK 5.0
4. IBM pSeries running Red Hat Enterprise Linux 4 (Nahant Update 4) PPC64 using IBM SDK 5.0
5. IBM pSeries running AIX 5L 5.2 PPC32 using IBM SDK 5.0
6. IBM pSeries running AIX 5L 5.2 PPC64 using IBM SDK 5.0
7. IBM zSeries running SUSE Linux Enterprise Server s390x using IBM SDK 5.0
8. Sun Solaris 8 using Sun JDK 5.0, 6.0
9. Sun Solaris 9 using Sun JDK 5.0, 6.0
10. Sun Solaris 10 using Sun JDK 5.0, 6.0

All of these platforms use the same cross-platform `.jar` binary. IBM affirms that the module operates correctly and maintains its FIPS compliance on these platforms.

2 Cryptographic Module Specification

The IBM CryptoLite for Java 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 CLiJ module running in the following configurations:

1. Windows Vista Ultimate x86 using Sun JDK 6.0.

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

1. the tested distribution (`jar`) file had SHA-256 hash:
   `aedf dd06 f63e c5c5 e86c eadb aa21 6ed7 fe3b 4b74 a37a 972c 57c2 93f3 ad37 f5a7`
2. and a SHA-1 hash of
   `6fdc aca6 fe15 42a9 591b c641 de0c 6a27 26e8 5e48`

Note that the Java binary is cross-platform. The same binary has been tested by IBM in all validation configurations.

The module running on the above platforms was validated as meeting all **FIPS Pub 140–2** Level 1 security requirements. The CLiJ module is packaged in a single `jar` distribution, `c4clij.jar`, which contains all the code for the module. The library, being a `jar` file, is self-contained. (Other non-functional support files, such as supporting documentation, may also be included in the distribution.)

Supported Algorithms The following algorithms are supported by the CLiJ module:

- Signature generation / verification using ECDSA*
- Key agreement using ECDH*
- Hashing using SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512
- Encryption using AES with 128, 192 and 256 bit keys.
- Random number generation using DRNG.

*Supported parameters are listed below.

Supported Elliptic Curve Parameters The CLiJ module supports the following parameters. Note that these only include EC over prime fields and not EC over binary fields.

Key length	OID	ANSI X9.62	NIST	SEC2
192 bits	1.2.840.10045.3.1.1	prime192v1	P-192	secp192r1
224 bits	1.3.132.0.33	N/A	P-224	secp224r1
256 bits	1.2.840.10045.3.1.7	prime256v1	P-256	secp256r1
384 bits	1.3.132.0.34	N/A	P-384	secp384r1
521 bits	1.3.132.0.35	N/A	P-521	secp521r1

Security level This document describes the security policy for the IBM CryptoLite for Java with Level 1 overall security as defined in **FIPS Pub 140-2**[4].

Module components

Type	Name	Release	Date	SHA-256 hash
Java 5 or above				
Software (`jar`)	`c4clij.jar`	4.2.FIPS	2007.10.26	see above
Documentation	CLiJ User Guide	4.2.FIPS	2007.10.26	N/A

Module boundary The CLiJ API represents the logical boundary of the module. The physical cryptographic boundary for module is defined as the enclosure of the host on which the cryptographic module is to be executed (see Figure 1). Layers between the logical and the physical boundary are the Java Virtual Machine on which the CLiJ module executes, the operating system of the host and the hardware components of the host.

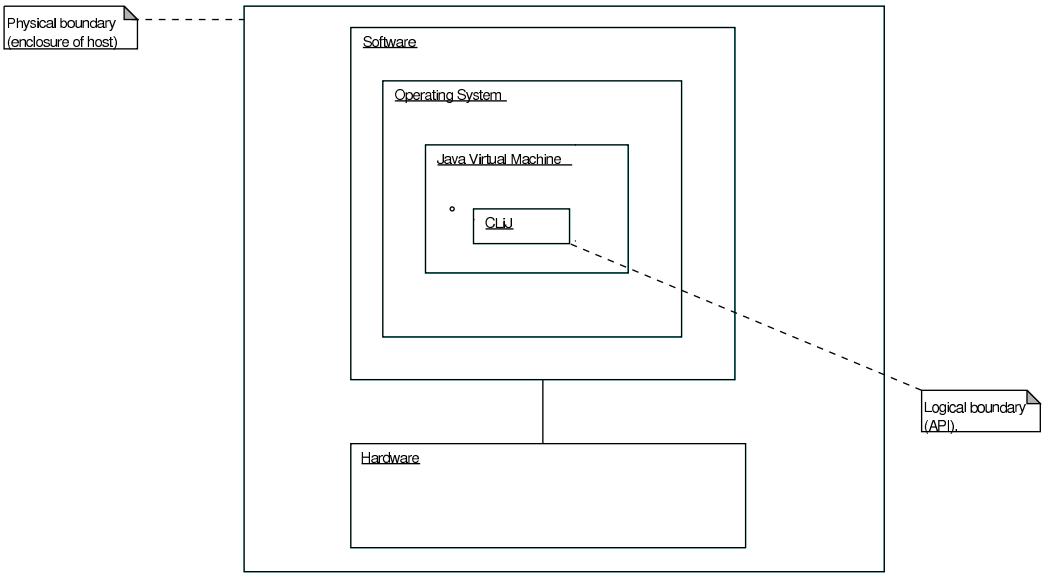

Figure 1: Module boundary.

3 Cryptographic Module Security Level

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

Security Requirements Section	Level
Cryptographic Module Specification	1
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 Java are not meaningful for the library itself. System utilizing CLiJ 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 *CLiJ physical interfaces* are the boundaries of the host running CLiJ library code. The underlying *logical interface of the module is Java JCE Application Program Interface (API)* [5]. JCE users access the JCE interface, and may not utilize underlying functions directly.

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. The public API is a standard JCE/JCA interface, allowing standardized access to queries and services. (Lower-level APIs are not directly accessible through JCE services.)

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. (Java memory management enforces the correctness of memory management, without involving library code.)

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 Java programs using the same method as other JCE providers are accessed. The module interface specification publishes the list of services, which in turn is made available to JCE-aware applications. (Access is restricted to functions published in the the JCE interface.)

Module Status The CLiJ communicates any error status asynchronously through the use of its documented return codes (i.e., exceptions). It is the responsibility of the calling application to handle exceptions in a FIPS 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 are reported as an unexpected error condition and terminates the CLiJ instance. These exceptions provide system-level failure notification, not just CLiJ 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

Authorized service	Officer	User
Officer services		
Invoke FIPS self-tests	Yes	Yes
Installation of Module	Yes	No
User services		
AES encryption/decryption	No	Yes
EC signature generation and verification **(ECDSA)**	No	Yes
EC key generation	No	Yes
EC Diffie-Hellmann (ECDH) key agreement	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
DRNG, obtain random number (FIPS 186-2/ANSI X9.31 generator)	No	Yes
Auxiliary functions	No	Yes

Table 3: Services by role

5.2 Services

The module provides *queries* and *commands* (Table 5 and 4). 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 5.

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

Service	Notes	Modes	Approved?	Role	
				Officer	User
Symmetric encryption and decryption					
AES	128, 192, or 256 bit keys (FIPS 197)	ECB, CBC	**yes**	no	yes
Public-key algorithms					
ECDSA sign/verify	NIST curves P-192 to P-521	sign/verify	**yes**	no	yes
EC key generation	NIST curves P-192 to P-521	N/A	**yes**	no	yes
ECDH key agreement	NIST curves P-192 to P-521	N/A	**yes**	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
Random number generation					
DRNG	FIPS 186-2, ANSI X9.31	N/A	**yes**	no	yes

Table 4: Commands, grouped by functionality

Service	Notes	Role	
		Officer	User
Module status			
Query mode	always in FIPS mode		
Integrity checks			
Power-up test	automatic before first use; includes binary integrity check	(yes)	no
Self-tests	functional KATs	yes	yes
Operational correctness checks			
RNG tests	continuously performed (automatic)	N/A	N/A
Comprehensive test application			
`selftest` application	very high coverage (*external utility*)	(yes)	(yes)

Table 5: 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 6: EC curve support

All algorithms support all combinations of key sizes and modes, where applicable. All supported algorithms support only approved modes.

Elliptic curve (EC) support is limited to curves over prime fields, using the curves in Table 6. The supported NIST P-curves have corresponding SECG or ANSI equivalents [1, 2, 3].

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

6 Operational Environment

The CLiJ security module is written in Java. Extensive internal consistency checks verify both user input and library configuration, terminating early if errors are encountered. Memory management is enforced by the Java runtime, below the library level, and this protection may not be bypassed by the library or its callers.

The module implements only 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 keys are used in a FIPS compliant mode, and approved modes are not combined to non-approved compounds.

CLiJ is developed and maintained according to IBM's 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 CLiJ library does not provide internal long-term cryptographic key storage; persistent storage management is assisted by the host JVM. 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 the 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 the FIPS-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. Key generation algorithms use the DRNG engine seeded with 20 bytes of true random data. Seed is derived internally, and may not be influenced through standard APIs.

Key Establishment Predefined EC Diffie-Hellmann (DH) key establishment curves are used, providing *key establishment strength from 80 to 256 bits*. Standard NIST P-curves are available for ECDH (see Table 6).

Key Protection *To enforce compliance with FIPS 140–2 key management requirements on the CLiJ library itself, code issuing CLiJ 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 CLiJ module in the clear.*

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 CLiJ module.

Key Destruction Released objects are destroyed by a custom `finalize` method, wiping memory before returning the region for garbage collection. Other key destruction activities, such as announcing revocation of an asymmetric key, are outside the scope of library activities.

6.2 Physical Security

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

6.3 EMI/EMC

EMI/EMC properties of the CLiJ deployment are identical to those of the host server or client. The library itself has no EMI/EMC properties.

7 Self-tests

The CLiJ 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 public-key keypairs.

Startup Self-Tests "Power-up" self-tests are performed automatically when the CLiJ 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 CLiJ 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-1-based HMAC of the all of the module `jar` file. Initialization will only succeed if this HMAC is valid. The HMAC field is prepared during `jar` file generation, during the last step of building. (Integrity verification is internal and not exposed through the JCE interface.)

The module tests the following cryptographic algorithms: AES, EC (sign/verify, key agreement, key generation), ECDH (KAT and random value test), HMAC (KAT), SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, and the DRNG.

Selftests are performed in logical order, verifying library integrity incrementally. SHA-1 and HMAC/SHA-1 selftests are executed before the library signature is verified.

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 and all data output from the responsible function during the error condition is inhibited. 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. (Note that the TRNG is not directly accessible through the JCE API, but this detail is relevant to understand DRNG failures.)

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 CLiJ 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 may not invoke KATs; obviously, reloading the library forces a new retest.

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 CLiJ should verify the SHA-256 hash of the executable image.

  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 CLiJ is present. (For legacy systems without a `sha256sum` utility, a SHA-1 hash is provided for `sha1sum` use.)

- Applications and libraries using CLiJ 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 CLiJ 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 by IBM during validation.)

  The module must not be used by multiple callers simultaneously such that they may interfere with each other. Note that since CLiJ 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., each object's ownership is uniquely determined).

- Applications using CLiJ 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 CLiJ library itself, just the application providing the keys to CLiJ. It is noted here for the sake of completeness.

- Applications utilizing CLiJ services must avoid combining approved algorithms or modes of operation to non-approved compound operations. (This requirement may not be enforced by the library itself, it depends on caller cooperation.)

- To be in FIPS mode, the CLiJ 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 supports only approved algorithms and modes of operation. After it passes initial selftests, it is in FIPS mode, and never leaves FIPS mode.

Applications utilizing CLiJ services must enforce key management compliant with FIPS 140–2 requirements. This should be indicated in an application-specific way that's directly observable by administrators and end-users. The application must not combine approved algorithms and operations to non-approved compounds. (While such application-specific details are outside the scope of the CLiJ validation, they are mentioned here for completeness.)

8.3 Testing/Physical Security Inspection Recommendations

In addition to automatic tests, described elsewhere in this document, CLiJ users may invoke FIPS mode by reinitializing their library instance. *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).

Generated binaries (`c4clij.jar`) is exercised by a set of maximal-coverage selftests as part of the build process as a Quality Assurance step. The test suite provides extensive test coverage of the generated library, beyond **FIPS Pub 140–2** requirements. The release test suite is time and processor intensive, and it is not shipped with the generated binary, it only verifies correctness during building.

Apart from prudent security practice of server applications, no further restrictions are placed on hosts utilizing CLiJ services.

9 Glossary

DRNG Deterministic Random Number Generator, a deterministic function expanding a "true random" seed to a pseudo random sequence.

JCA Java Cryptography Architecture

JCE Java Cryptography Extension. Interface definition standard that allows crypto providers to present a uniform interface to Java applications (registered and enumerated centrally). See also JCA.

KAT Known Answer Test

OS Operating System

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

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

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

[5] Sun Microsystems. *Java Cryptography Extension API Specification & Reference (version 1.2.2)*, June 2000.

Policy revision 35.