IBM CryptoLite for C
Version 3.23

(Non-proprietary)

Security Policy*

IBM Research
Zürich Research Laboratory

IBM Crypto Competence Center
Copenhagen

March 29, 2007

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

*Policy revision 100

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 (3.23). 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. The library name is `libclic.a` on AIX systems. Binaries contain 32 and 64-bit versions of the shared library, and the operating system selects the variant to load.

This validation specifically targeted the AIX build.

The CLiC 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.

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 System p configurations:

1. an 44P Model 270 host running AIX 5200-07 (32-bit),
2. an 44P Model 270 host running AIX 5200-07 (64-bit),
3. an 44P Model 270 host running AIX 5300-03 (32-bit),
4. an 44P Model 270 host running AIX 5300-03 (64-bit),

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

1. the tested AIX `.a` file had SHA-256 hash: `e2e7 66ad 918b deea e755 31c6 d441 0a0b b625 dd7f fa83 566d 63bc 1ad5 b3c2 0d8b`
2. and a SHA-1 hash of `da81 106c 9c47 c849 e047 ad46 658c 9cae f760 7ba1`

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.) The deployed DLL name is `libclic.a` on all AIX variants.

The CLiC library also runs on many other platforms, including other AIX versions, MVS and USS32 (on mainframes), other Windows variants, HP-UX, Sun/Solaris, and PalmOS. However, the IBM CryptoLite for C module was not tested on these platforms as part of this validation effort.

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**[2].

Module components

Type	Name	Release	Date	SHA-256 hash
AIX 5200-07 and 5300-03				
Software (DLL)	`libclic.a`	3.23	2006.06.01	see above
Documentation	CLiC User Guide	3.23	2006.06.01	N/A

Vendor-affirmed testing was performed on one platform with a Java-less DLL (MVS). The CLiC API, documentation etc. is identical to the AIX version.

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	(1)
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.

In systems with Java support, a Java interface may be provided over the CLiC API, enabling one to access CLiC functionality from Java applications. This interface simply transforms data, and provides no encryption or security functionality of its own.

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.

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 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.

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 labeled "*legacy*" are supported only for backwards compatibility with existing applications.

Single DES is labeled as non-approved, but its usage is deprecated, except for legacy compatibility purposes. A dedicated build option enables CLiC builds to specifically inhibit single-DES functionality. The module under validation did not enable this switch, it still allows single-DES operations (flagging them as non-approved).

Format conversions, labeled 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.

Authorized service	**Officer**	**User**
Officer services		
Invoke FIPS self-tests	Yes	Yes
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
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
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
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
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, (CTR), CCM, (GCM), (CTS)	**yes**	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
Public-key algorithms					
DSA key/prm generation	up to 1024 bit modulus (FIPS 186 key sizes)	N/A	**yes**	no	yes
DSA sign/verify	FIPS 186 key sizes	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
Diffie-Hellmann (DH)	up to 2048 bits modulus	key agreement	**yes**	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
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					
Force library to non-FIPS mode	can not return to FIPS mode	N/A	N/A	yes	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_isFIPSmode`)	yes	yes
Integrity checks			
Power-up test	automatic before first use includes `CLiC_verifyDLLIntegrity`	(yes)	no
Self-tests	`CLiC_fips140SelfTests`	yes	yes
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)	(yes)

Table 6: Queries

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 that 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 outsize 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 [1].* 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, starting at 512 bits, up to 2048 bit modulus. Therefore, *DH key establishment provides from 56 to 112 bits of encryption strength.*

RSA-based key establishment has no fixed upper limit on modulus size.

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 in 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.

A dedicated CLiC service provides a convenient aid for key destruction, a "wipe" function to portably clear key structures from working memory (`CLiC_memset`). This is the preferred method of key destruction, as it is never bypassed by compiler optimizations (unlike regularly used applications of `memset` from the standard library).

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 all of the module binary. Initialization will only succeed if this HMAC is valid. The HMAC field is prepared during shared library (DLL) generation, during the last step of building. (Integrity verification is contained in `CLiC_verifyDLLIntegrity`.)

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

Selftests 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 first 32-bit output block generated by the DRNG is never used for any purpose other than initiating the continuous DRNG test which compares every newly generated 32-bit RN block with the previously generated block. The test fails if the DRNG outputs the same 32-bit value twice subsequently.

If the DRNG outputs identical, subsequent 32-bit 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 32-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 became operational, one can always invoke the `CLiC_fips140SelfTests` 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_isFIPSmode` 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. *If users of CLiC perform non-FIPS-compliant operations, they must indicate to the library that it is no longer in FIPS mode* (through the `CLiC_setNonFIPSmode` call).

  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_isFIPSmode` 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. If the application performs non-approved operations outside the module, it may force the module to non-FIPS mode through a function call (`CLiC_setNonFIPSmode`). As intended, such a call effectively sets the whole infrastructure based on CLiC to non-FIPS mode.

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 outsize 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_verifyDLLIntegrity`), 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.

9 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 pseudo-random sequence.

KAT Known Answer Test

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

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

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

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

Policy revision 100, last archived: 2007.03.29. 13:22:26, check-in by user *tvi (Tamas Visegrady)*.