IBM Java JCE FIPS 140-2 Cryptographic Module

Security Policy

IBM JAVA JCE FIPS 140-2 Cryptographic module
Revision: 1.1

January 15, 2004

Status: Final

First Edition (October 2003)
This edition applies to the First Edition of the IBMJCEFIPS – Security Policy and to all subsequent versions until otherwise indicated in new editions.

Table of Contents

Introduction

The IBM® Java® JCE (Java Cryptographic Extension) FIPS provider (IBMJCEFIPS) for Multi-platforms is a scalable, multi-purpose cryptographic module that supports only FIPS approved cryptographic operations via the Java2 Application Programming Interfaces (APIs). The IBM Java JCE FIPS provider (hereafter referred to as IBMJCEFIPS) comprises the following Federal Information Processing Standards (FIPS) 140-2 [Level 1] compliant components:

- IBMJCEFIPS for Solaris®, Windows®, AIX®, z/OS®, AS/400®, Linux® (Red Hat and SuSE®)

In order to meet the requirements set forth in the FIPS publication 140-2, the encryption algorithms utilized by the IBMJCEFIPS provider are isolated into the IBMJCEFIPS provider cryptographic module (hereafter referred to as cryptographic module), which is accessed by the product code via the Java JCE framework APIs. As the IBMJCEFIPS provider utilizes the cryptographic module in an approved manner, the product complies with the FIPS 140-2 requirements when properly configured.

This document focuses on the features and security policy provided by the cryptographic module, and describes how the module is designed to meet FIPS 140-2 compliance.

Operation of the Cryptographic Module

The cryptographic module must be utilized in a compliant manner, as described herein, to maintain FIPS 140-2 compliance. It is the application and application administrator's responsibility to understand and deploy the proper configuration for compliance.

The module is available as a software module on multiple platforms. The platforms tested are outlined in the *Cryptographic Module Specification* section of this document. The module must be used in one of the specified environments.

An application utilizes the module through the interfaces specified in the *Cryptographic Module Interfaces* section of this document. A list of the basic services provided through these interfaces may be found in the *Cryptographic Module Services* section of this document. A complete list of all services and details on their usage can be found in the *IBM Java JCE FIPS (IBMJCEFIPS) Cryptographic Module API* Javadoc.

The module provides for two operator roles:

- Crypto Officer
- User

There is no maintenance role in this cryptographic module.

An application must use the IBMJCEFIPS provider to enable the use of appropriate cryptographic functions in a FIPS approved manner. The application calling the IBMJCEFIPS provider must understand the roles of the APIs, Crypto Officer vs User. The *Cryptographic Module Roles* section of this document details the APIs that apply to each role. This cryptographic module contains only FIPS compliant operations, there are no special actions needed or limited cryptographic operations to instantiate FIPS mode.

The module can provide for protection of sensitive data, such as keys or contexts. Information on key protection is outlined in the C*ryptographic Module Key Management* section. When the module is initialized, it validates its own integrity, and verifies the algorithms are functioning correctly. The *Cryptographic Module Self-Tests* section details the internal tests performed by the module.

The module’s physical security relies on the physical security of the computer. Steps to deploy and maintain this secure environment are outlined in the *User Guidance* section of this document.

Cryptographic Module Specification

The cryptographic module is a software module, implemented as a Java archive (JAR). The software module is accessible from Java language programs through an application program interface (API). Some of the available API functions are listed below in the *Cryptographic Module Services* section. Usage guidelines and details of the full API function set are available in the *IBM Java JCE FIPS (IBMJCEFIPS) Cryptographic Module API* Javadoc.

The module is validated to the following FIPS 140-2 defined levels:

Overall	**Security Level 1**
Cryptographic Module Specification	Security Level 1
Cryptographic Module Ports and Interfaces	Security Level 1
Roles, Services, and Authentication	Security Level 1
Finite State Model	Security Level 1
Physical Security	Security Level 1
Operational Environment	Security Level 1
Cryptographic Key Management	Security Level 1
EMI/EMC	Security Level 1
Self-Tests	Security Level 1
Design Assurance	Security Level 1
Mitigation of Other Attacks	Security Level 1

As outlined in section G.5 of the Implementation Guidance for FIPS 140-2, the module maintains its compliance on other operating systems, provided:

- The GPC uses the specified single user operating system/mode specified on the validation certificate, or another compatible single user operating system, and
- The source code of the software cryptographic module does not require modification prior to recompilation to allow porting to another compatible single user operating system.

The IBMJCEFIPS provider was tested on a machine running Microsoft Windows 2000® Advanced Server with Service Pack 4 with JVM 1.4.1 and Microsoft Windows 2000® with Service Pack 4 operating system with JVM 1.3.1 03. The software module maintains compliance when running on the Microsoft Windows 95, ® Microsoft Windows 98®, Microsoft Windows Me®, Microsoft Windows NT®, Microsoft Windows 2000®, and Microsoft Windows XP® operating systems, as well as, JVMs at the 1.4.x level on those operating systems.

The IBMJCEFIPS provider was tested on a machine running the AIX® 5.2 operating system with JVM 1.3.1 and JVM 1.4.1. The software module maintains compliance when running on other versions of AIX®, as well as, JVMs at the 1.4.x level on those AIX® versions.

The IBMJCEFIPS provider was tested on a machine running the Solaris® 5.8 operating system with JVM 1.3.1 and JVM 1.4.1. The software module maintains compliance when running on other UNIX based operating systems, such as HP-UX, as well as, JVMs at the 1.4.x level on those operating systems.

The IBMJCEFIPS provider was tested on a machine running the Red Hat Linux AS 2.1 operating system with JVM 1.4.1. The software module maintains compliance when running on other Linux based operating systems, as well as, JVMs at the 1.3.1 or 1.4.x level on those operating systems.

The IBMJCEFIPS provider was tested on a machine running the SuSE® Linux SLES 8.0 operating system with JVM 1.4.1. The software module maintains compliance when running on other Linux based operating systems, as well as, JVMs at the 1.3.1 or 1.4.x level on those operating systems.

The IBMJCEFIPS provider was tested on a machine running the z/OS® R1V4 operating system with JVM 1.4.1. The software module maintains compliance when running on other z/OS operating system releases, as well as, JVMs at the 1.3.1 or 1.4.x level on those operating system releases.

The IBMJCEFIPS provider was tested on a machine running the IBM Operating System/400 V5R2M0 operating system with JVM 1.4.1. The software module maintains compliance when running on other IBM Operating System/400 operating system releases, as well as, JVMs at the 1.3.1 or 1.4.x level on those operating system releases.

The module supports the following approved algorithms:

Type	**Algorithm**	**Specification**
Symmetric Cipher	AES (ECB, CBC, OFB, CFB and PCBC)	FIPS 197
	DES (ECB, CBC, OFB, CFB and PCBC) – For legacy systems only	FIPS 46-3
	Triple DES (ECB, CBC, OFB, CFB and PCBC)	
Message Digest	SHA1	FIPS 180-1
	HMAC-SHA1	FIPS 198
Asymmetric Cipher	RSA	PKCS #1
Key Agreement	Diffie-Hellman	PKCS #3 (Allowed in Approved mode)
Random Number Generation	X 9.31 PRNG	ANSI X 9.31 1998
	FIPS 186-2 Appendix 3.1	FIPS 186-2
Digital Signature	DSA (512 - 1024)	FIPS 186-2
Digital Signature	RSA (512 – 2048)	FIPS 186-2

In addition, the module supports the following non-approved algorithms:

Type	**Algorithm**	**Specification**
Random Number Generation	Universal Software Based Random Number Generator	Available upon request from IBM. Patented by IBM, EC Pat. No. EP1081591A2, U.S. pat. Pend.

Cryptographic Module Interfaces

The cryptographic physical boundary is defined at the perimeter of the computer system enclosure on which the cryptographic module is to be executed, and includes all the hardware components within the enclosure. The cryptographic module interfaces with the Central Processing Unit (CPU) of the respective platform. The RAM and hard disk found on the computer are memory devices that store and execute the cryptographic module and its data.

The cryptographic module is classified as a “multi-chip standalone module” for FIPS 140-2 purposes. Thus, the module’s physical interfaces consist of those found as part of the computer’s hardware, such as the keyboard, mouse, disk drive, CD drive, network adapters, serial and USB ports, monitor, speakers, etc. The module’s logical interface is provided through the documented API.

Each of the FIPS 140-2 defined logical interfaces are implemented as follows:

- Data Input Interface – variables passed in with the API function calls
- Data Output Interface – variables passed back with the API function calls
- Control Input Interface – the API function calls exported from the module
- Status Output Interface – return values and error exceptions provided with the API method calls

Cryptographic Module Services

The module services are accessible from Java language programs through an Application Program Interface (API). The application will be required to call the IBMJCEFIPS provider (as opposed to another JCE provider) through the normal Java 2 mechanisms such as specifically adding the provider name to the getInstance call as part of the instantiation of a cryptographic object or by placing the IBMJCEFIPS provider higher in the provider list and allowing the JVM to select the first provider that has the requested cryptographic capability. Usage guidelines and details of the API function are available in the *IBM Java JCE FIPS (IBMJCEFIPS) Cryptographic Module API* document.

The following is a high level description of the basic capabilities available in the cryptographic module (all services are for the user role unless otherwise noted).

This is intended to outline the basic services available in the cryptographic module to allow a determination as to whether these services will adequately address the security needs of an application. Usage guidelines and details of all of the API functions are available in the *IBM Java JCE FIPS (IBMJCEFIPS) Cryptographic Module API* document.

Self Test

This section describes some of the capabilities that are available as they relate to the self test the cryptographic module performs to validate its own integrity and to verify the algorithms are functionally correct.

Services	**Description**
IsSelfTestInProgress	Identifies if a self test is currently in progress. Call is based on a SelfTest object returned from the getSelfTest call.
GetSelfTestFailure	Returns the exception associated with the self test failure or null if no failure was encountered. Call is based on a SelfTest object returned from the getSelfTest call.
RunSelfTest	Performs the known answer self tests. Call is based on a SelfTest object returned from the getSelfTest call. **This is a Cryptographic Officer role, call.**
IsFipsRunnable	Identifies if the crypto module is runnable, has completed self test with no errors, and is in "Ready" state. Call is based on a SelfTest object returned from the getSelfTest call.
IsFipsCertified	Identifies if the cryptographic module is FIPS 140-2 validated. Call is based on a provider object.
GetFipsLevel	Returns the FIPS 140-2 validation level of the cryptographic module. Call is based on a provider object.
GetSelfTest	Returns a SelfTest object that can be used to execute any of the SelfTest class methods. Call is based on a provider object.
IsFipsApproved	Identifies if the cryptographic operation is FIPS 140-2 validated. Call is based on a

	cryptographic object.

Data Encryption/Decryption and Hashing (Digest)
This section describes some of the capabilities that are available as they relate to encryption/decryption (Cipher) of data and digesting or hashing (MessageDigest) of data.

Services	**Description**
getInstance Cipher.getInstance MessageDigest.getInstanc e	Creates a cryptographic object (Cipher/MessageDigest) for a selected algorithm. Also used to select the cryptographic provider to be used by that object. Cipher allows for DES, 3DES, and AES algorithms with various cipher modes and paddings. MessageDigest allows for SHA-1 hashing.
Init Cipher.init MessageDigest.init	Intitializes the cryptographic object for use. This includes the mode (encryption or decryption) and the cryptographic key. This call is based on a cryptographic object.
Update Cipher.update MessageDigest.uipdate	Updates the cryptographic object with data to be encrypted/decrypted. This call is based on a cryptographic object.
doFinal Cipher.doFinal MessageDigest.doFinal	Updates the cryptographic object with data to be encrypted/decrypted and returns the data in encrypted or decrypted form (based on the init). This call is based on a cryptographic object

Key Generation
This section describes some of the capabilities that are available as they relate to keys.

Services	**Descritption**

getInstance KeyGenerator.getInstance	Creates a cryptographic object (KeyGenerator) for a selected algorithm. Also used to select the cryptographic provider to be used by that object.
Init	Intitializes the cryptographic object for use. This call is based on a cryptographic object.
GenerateKey	Generates a cryptographic key. This call is based on a cryptographic object.

Services	**Description**
getInstance KeyPairGenerator.getInsta nce	Creates a cryptographic object (KeyPairGenerator) for a selected algorithm. Also used to select the cryptographic provider to be used by that object.
initialize	Intitializes the cryptographic object for use. This call is based on a cryptographic object.
generateKeyPair	Generates a cryptographic key pair. This call is based on a cryptographic object.

Key Security
In accordance with the FIPS 140-2 standards this cryptographic module provides the user of keys the ability to zero out the key information via a new API.

Service	**Description**
(crypto key object). zeroize	Zeros out the key(s) associated with a cryptographic object. This call is based on a cryptographic object.

Signature
This section describes some of the capabilities that are available as they relate to signature generation and verification.

Service	Description
getInstance Signature.getInstance	Creates a cryptographic object (Signature) for a selected algorithm. Also used to select the cryptographic provider to be used by that object.
InitSign	Intitializes the cryptographic object for use. This includes the cryptographic private key. This call is based on a cryptographic object.
Update	Update a byte or byte array in the data to be signed or verified. This call is based on a cryptographic object.
Sign	Get message digest for all the data thus far updated, then sign the message digest. This call is based on a cryptographic object.
InitVerify	Intitializes the cryptographic object for use. This includes the cryptographic public key. This call is based on a cryptographic object.
verify	Verify the signature (compare the result with the message digest). This call is based on a cryptographic object.

Secret Key Factory
This section describes some of the capabilities that are available as they relate to symmetric keys.

Service	Description
GetInstance	Creates a cryptographic object (SecretKeyFactory) for a selected algorithm. Also used to select the cryptographic provider to be used by that object.
GetKeySpec	Returns a specification (key material) of the given key in the requested format.
generateSecret	Generates a SecretKey object from the provided key specification (key material).

KeyFactory
This section describes some of the capabilities that are available as they relate to asymmetric keys.

GetInstance	Creates a cryptographic object (KeyFactory) for a selected algorithm. Also used to select the cryptographic provider to be used by that object
GeneratePublic	Generates a public key object from the provided key specification (key material).
GeneratePrivate	Generates a private key object from the provided key specification (key material).
getKeySpec	Returns a specification (key material) of the given key object in the requested format.

Cryptographic Module Roles

The cryptographic module implements both a Crypto Officer and a User role, meeting all FIPS 140-2 level 1 requirements for roles and services. A Maintenance Role is not implemented.

Cryptographic Officer role

The Crypto Officer role has responsibility for initiating on-demand self test diagnostics. This is accomplished through the runSelfTest API call described in the *IBMJCEFIPS provider Cryptographic Module API* document.

Cryptographic User role

The User role has the responsibility for operating cryptographic functions on data.

User guidance information is available in the *IBMJCEFIPS provider Cryptographic Module API* document.

There is no maintenance role.

Only one role is implicitly active in the module at a time.

Cryptographic Module Key Management

The module supports the use of the following cryptographic keys: Diffie-Hellman public/private keys, DES, Triple DES, AES, RSA public/private keys, DSA public/private keys, and HMAC SHA1.

Operators of the module have full access to key material. These keys are accessed by calling the various cryptographic services specified in the *IBMJCEFIPS provider Cryptographic Module API* document.

Key Generation

Symmetric keys are generated using the X9.31 pseudo random-number generation algorithm.

DSA parameters, along with public and private keys are generated using the random number algorithms as defined in FIPS 186-2. DSA and RSA key pairs are generated as defined in FIPS 186-2.

IBM has invented a scheme to generate randomness on a wide range of computer systems. The patented scheme, called the Universal Software Based True Random Number Generator, utilizes random events influenced by concurrent activities in the system (e.g. interrupts, process scheduling, etc). The run time of the algorithm will vary depending of the state of the system at the time of seed generation, and will be dependent on the type of system. The Universal Software Based True Random Number Generator is used to create a random seed value that is used in the PRNG algorithms.

Key Storage

We do not support key storage within the IBMJCEFIPS cryptographic module.

Key Protection

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 central processing unit (CPU) hardware control access to that space.

Each instance of the cryptographic module is self-contained within a process space. Only one instance of the module is available in each process space. All keys are associated with the User role.

Key Zeroization

All cryptographic keys and contexts are zeroized when an operator:

- Disposes of a key using the zeroize API call for that key object.
- When Java garbage collection is performed for an object no longer referenced, as part of the objects finalize method.
- Powers off the module by unloading it from memory

Cryptographic Module Self-Tests

When an application references the cryptographic module within the JVM in its process space, an initialization routine is called by the JVM before control is handed to the application. This initialization route automatically executes the power up tests to ensure correct operation of the cryptographic algorithms.

The integrity of the module is verified by performing a HMAC validation of the cryptographic module’s classes contained in the module’s jar file. The initialization route will only succeed if the HMAC is valid.

Self-tests include known answer tests for the RSA, Diffie-Hellman, SHA1, DES, Triple DES, AES, RSA, DSA, HMAC SHA1 cryptographic algorithms and pseudo random number generation. Should any self-test fail, the module transitions to the Error state.

Additionally, conditional tests are performed when symmetric or asymmetric keys are generated. These tests include a continuous random number generator test and pair-wise consistency tests of the generated DSA and RSA keys.

These self tests can also be run on demand by the cryptographic officer via the runSelfTest method.

User Guidance

Programming practices

This section contains guidance for application programmers to avoid practices that could potentially compromise the secure use of this cryptographic module.

- Zeroize - the zeroize method should be used when a cryptographic key object is no longer needed to remove the key from memory. While normal Java garbage collection will zeroize the key from memory as part of the object finalizer method it is a safer coding practice to explicitly call the zeroize method when an application is finished with a key object.
- Statics – To ensure that each cryptographic object is unique and accessible only by the individual user it is important not to use static objects, as all users of the JVM share these objects.

  As the Java architecture creates objects that are unique to the application and this allows for “single” user access to the cryptographic operations and data it is recommended that an application not create static objects. Static objects are shared in the Java architecture and the creation of a static object would be counter to the unique object method of controlling access and data.

- An application that wishes to use FIPS validated cryptography must use the IBM Secure random algorithm associated with the IBMJCEFIPS provider for the source of random data needed by some algorithms.
- RSA Cryptographic Cipher may only be used to Encrypt and Decrypt keys for transport to stay within the boundaries of the Approved Mode of FIPS 140-2 Level 1.
- One way to help alleviate performance problems is by creating a single source of randomness (IBMSecureRandom) and using that object when ever possible.

Installation and Security rules for using IBMJCEFIPS

This section contains guidance for the installation and use of the FIPS 140-2 level 1 cryptographic module.

The IBMJCEFIPS provider jar file must be accessible via the Java CLASSPATH and should be installed in the directory lib/ext as this is a secure location and is also automatically available via the JVM without a CLASSPATH update.

The application will be required to call the IBMJCEFIPS provider (as opposed to another JCE provider) through the normal Java 2 mechanisms such as specifically adding the provider name to the getInstance call as part of the instantiation of a cryptographic object or by placing the IBMJCEFIPS provider higher in the provider list and allowing the JVM to select the first provider that has the requested cryptographic capability.

Cryptographic Module Operating system environment

Framework

The cryptographic module is dependant on the operating system environment being set up in accordance with FIPS 140-2 specifications. For this cryptographic provider a valid commercial grade installation of a Java SDK 1.3.1 or higher JVM must be available.

A valid commercial grade installation of a Java SDK 1.3.1 or higher JVM that includes the Java Cryptographic Extension framework (Version 1.2.1) is required. (Please note that a JVM at 1.4.0 or higher already contains the JCE framework). In addition to the SDK and the JCE framework the IBMJCEFIPS provider is required.

The following is a brief overview of the JCE framework (A more detailed explanation of this framework is available at (http://java.sun.com/products/jce/doc/guide/HowToImplAProvider.html#MutualAuth)

In order to prevent unauthorized providers from plugging into the JCE 1.2.1 framework (herein referred to as "JCE 1.2.1"), and to assure authorized providers of the integrity and authenticity of the JCE 1.2.1 that they plug into, JCE 1.2.1 and its providers will engage in mutual authentication. Only providers that have authenticated JCE 1.2.1, and who in turn have been authenticated by JCE 1.2.1, will become usable in the JCE 1.2.1 environment. For more information about this, please see the above web page.

In addition, each provider does do self-integrity checking to ensure that the JAR file containing its code has not been tampered with. The JCE 1.2.1 framework is digitally signed. Providers that provide implementations for JCE 1.2.1 services must also be digitally signed. Authentication includes verification of those signatures and ensuring the signatures were generated by trusted entities. Certain Certificate Authorities are deemed to be "trusted" and any code signed using a certificate that can be traced up a certificate chain to a certificate for one of the trusted Certificate Authorities are considered trusted. Both JCE 1.2.1 and provider packages do embed within themselves the bytes for the certificates for the relevant trusted Certificate Authorities. At runtime, the embedded certificates will be used in determining whether or not code is authentic. Currently, there are two trusted Certification Authorities: Sun Microsystems' JCE Code Signing CA, and IBM JCE Code Signing CA.

In order to insure that an application is using the FIPS validated cryptographic module, the application is required to call the IBMJCEFIPS provider (as opposed to another JCE provider) through the normal Java 2 mechanisms such as specifically adding the provider name to the getInstance call as part of the instantiation of a cryptographic object or by placing the IBMJCEFIPS provider higher in the provider list and allowing the JVM to select the first provider that has the requested cryptographic capability.

Single user access (operating system requirements)

This cryptographic module adheres to the FIPS 140-2 level 1 requirement that the operating system must be restricted to a single operator mode (concurrent operators are explicitly excluded). The following explains how to configure a Unix system for single user. The general idea is the same across all Unix variants:

- Remove all login accounts except "root" (the superuser).
- Disable NIS and other name services for users and groups.
- Turn off all remote login, remote command execution, and file transfer daemons.

The Windows Operating Systems can be configured in single user mode by disabling all user accounts except the administrator. This can be done through the Computer Management window of the operating system. Additionally, the operating system must be configured to operate securely and to prevent remote login. This is accomplished by disabling all services (within the Administrative tools) that provide remote access (e.g. – ftp, telnet, ssh, and server) and disallowing multiple operators to log in at once.

Java object model

The use of Java objects within the cryptographic module. In Java each cryptographic object is unique. Thus when an application generates a cryptographic object for use that object is unique to that instance of the application. In this regard other processes have no access to that object and can therefore not interrupt or gain access to the information or activities contained within that object. In this way the cryptographic module protects the single users control of the cryptographic activities and data.
Further as the Self Test class is a Java static object there can be only one instance of that class in the JVM and that instance controls the Self Test activities. In other words if the Self Test fails, then no cryptographic objects for the IBMJCEFIPS provider in the JVM will be operational as the cryptographic module would be in “Error” state.

As the Java architecture creates objects that are unique to the application and this allows for “single” user access to the cryptographic operations and data. It is recommended that an application not create static objects. Static objects are shared in the Java architecture and the creation of a static object would be counter to the unique object method of controlling access and data.

Operating system restriction

The operation of the cryptographic module is assumed to be in single user mode in that only one user is on the system at any point in time.

Mitigation of other attacks

The IBMJCEFIPS provider has been obfuscated. Code obfuscation is provided by the commercial product KlassMaster. This level of optimized code makes it difficult to decompile and reuse the derived source code. IBM's test with popular de-compilers (e.g. Jasmine) has shown that de-compiled IBMJCEFIPS code for Java code cannot be compiled and used without extensive alteration

No other mitigation of other attacks is provided.

[1] National Institute of Standards and Technology. May 2001. *Security Requirements for Cryptographic Modules.* Federal Information Processing Standards Publication 140-2.

[2] National Institute of Standards and Technology. November 2001. *AES Key Wrap Specification.* Internet. 22 April 2002. http://csrc.nist.gov/encryption/kms/key-wrap.pdf

Notices