IBM Java JCE FIPS 140-2 Cryptographic Module

Security Policy

IBM JAVA JCE FIPS 140-2 Cryptographic Module
Revision: 1.7

Jan 2012

Status: Final

1.7 Edition (February 2012)
This edition applies to the 1.7 Edition of the IBMJCEFIPS – Security Policy and to all subsequent versions until otherwise indicated in new editions.

Table of Contents

1 Introduction

The IBM® Java® JCE (Java Cryptographic Extension) FIPS 140-2 Cryptographic Module (Version 1.7) for Multi-platforms is a scalable, multi-purpose cryptographic module that supports FIPS approved cryptographic operations via the Java2 Application Programming Interfaces (APIs). The IBM Java JCE FIPS 140-2 Cryptographic Module (hereafter referred to as IBMJCEFIPS) comprises the following Federal Information Processing Standards (FIPS) 140-2 [Level 1] compliant components:

- IBMJCEFIPS.jar

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 following all rules and guidelines in this Security Policy document.

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.

2 Operation of the Cryptographic Module

The cryptographic module must be utilized in a secure 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.

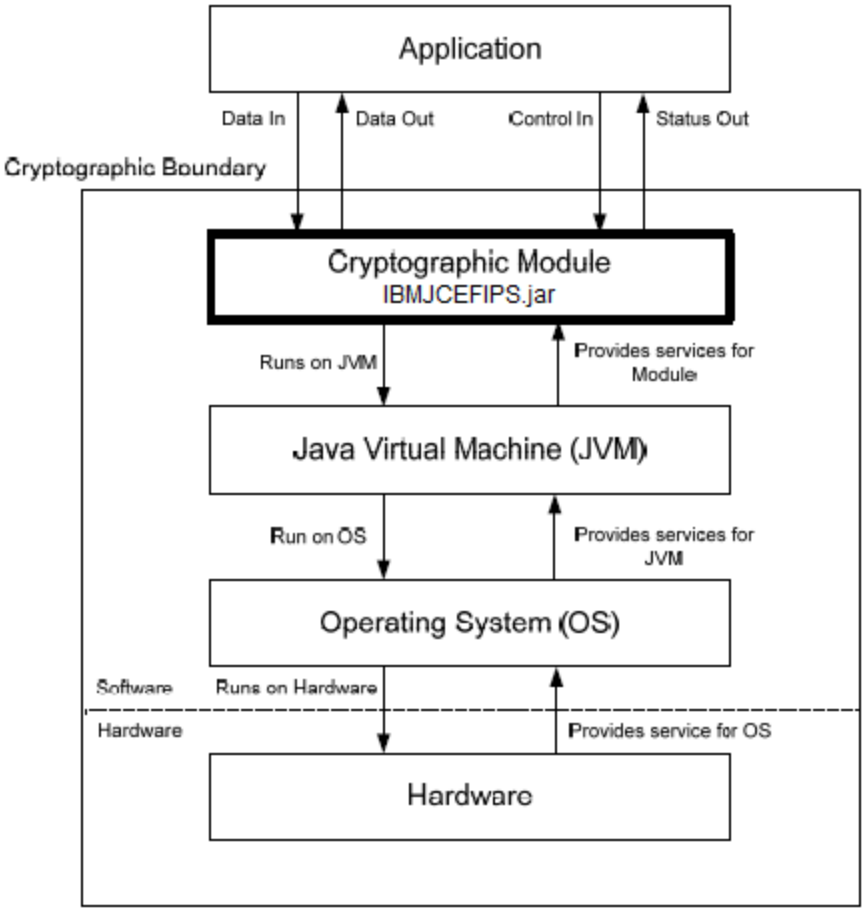

Figure 1: IBM Java JCE FIPS Logical Block Diagram

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 Crypto Officer and the User. The *Cryptographic Module Roles* section of this document details the APIs that apply to each role. In order to use the module in FIPS mode, the User must ensure that only FIPS Approved cryptographic algorithms are being invoked and/or algorithms are used in an approved manner.

The module can provide for protection of sensitive data, such as keys or cryptographic 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.

3 Changes from Version 1.3.1 to 1.7

The following was added to the 1.7 version of IBMJCEFIPS:

- Support for GCM mode in the AES cipher has been added (Non-Compliant)
- FIPS 186-3 DSA with SHA-256 signature support has been added
- Elliptic Curve support (over polynomial curves) for DSA signatures and DH key agreement has been added
- RSA key pair generation has been modified to have saner limits by default
- RSA and DSA have been modified to generate 2048 bit keys by default
- SHA-256 digests are used by default in signatures
- TLS 1.1 and 1.2 implementations added.
- SP 800-90A Hash-Based DRBG added.

4 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	N/A
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

5 Platforms Validated

The IBMJCEFIPS provider has been tested and is FIPS-validated when used on the following operating systems:

- Microsoft Windows 7 32-bit
- IBM AIX 7.1
- Solaris 11.0

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 7 operating system in single-user mode with JVM 1.6. The software module maintains compliance when running on the Microsoft Windows 95, ® Microsoft Windows 98®, Microsoft Windows Me®, Microsoft Windows NT®, Microsoft Windows 2000®, Microsoft Windows XP®, and Microsoft Windows Vista® operating systems and will be considered “vendor-affirmed” for such operating systems, as well as, JVMs at the 6.x level and 7.x level on those operating systems.

IBM performs testing on AIX, Solaris, HP, Red Hat Linux, SuSE® Linux, z/OS® and IBM Operating System/400 platforms and affirms IBMJCEFIPS operates correctly

on these platforms. These operating systems have not been explicitly validated, however, IBM is affirming its compliance as outlined in section G.5 of the Implementation Guidance for FIPS 140-2.

6 Cryptographic Algorithms Supported

The module supports the following approved algorithms:

Type	Algorithm	Specification	Certificate #
Symmetric Cipher	AES (ECB, CBC, OFB, and CFB modes)	FIPS 197	2107
	Triple-DES (ECB, CBC, OFB and CFB modes)	FIPS 46-3	1342
Message Digest	SHA1 SHA-256 SHA-384 SHA-512	FIPS 180-3	1830
Message Authentication	HMAC–SHA-1 HMAC–SHA-256 HMAC–SHA-384 HMAC–SHA-512	FIPS 198a	1281
Random Number Generation	FIPS 186-2 Appendix 3.1	FIPS 186-2	1082
DRBG	Hash-based DRBG	NIST SP 800-90A	228
Digital Signature	DSA (1024[1])	FIPS 186-2, FIPS 186-3	657
Digital Signature	RSA (1024[2] – 16384[3])	PKCS#1 v1.5 (FIPS 186-2 and FIPS 186-3)	1081

[1] The DSA implementation supports all key sizes from 512-2048, but only 1024 is allowed in approved mode.
[2] The RSA implementation also supports 512 bit key sizes, but this size may not be used in approved mode.
[3] Though IBM JCE FIPS supports key sizes up to 16384 bits, generating or working with keys of this magnitude is not recommended for all uses. Key sizes above 4096 bits are allowed for future-proofing and for specialized use-cases. Application developers using the IBM JCE FIPS jar should regulate the key sizes their applications are permitted to operate upon in order to ensure that the machine cannot be tied up

Digital Signature	ECDSA (192-521)	FIPS 186-2, FIPS 186-3	314

The module supports the following non-Approved cryptographic algorithms that are allowed for use in FIPS-mode:

Type	**Algorithm**	**Specification**
Message Digest	MD5	RFC 1321 (Allowed for use within the TLS protocol).
Asymmetric Cipher	RSA Key Transport	PKCS #1 with and without blinding (RSASSL) (Allowed in the Approved mode for key transport[4]) Provides between 80 and 256 bits of encryption strength
Key Agreement	Diffie-Hellman Shared Secret (256 –2048 bits)	PKCS #3 (Allowed in Approved mode[5]) Used with the TLS implementation Provides between 40 and 112 bits of encryption strength
Key Agreement	Elliptic-Curve Diffie-Hellman Shared Secret	Allowed in Approved mode. Used with the TLS implementation Provides between 80 and 256 bits of encryption strength
Digital Signature	DSAforSSL[6]	Allowed for use within the TLS protocol
Digital Signature	RSAforSSL[6]	Allowed for use within the TLS protocol
Random Number Generation	Universal Software Based Random Number Generator	Allowed in Approved mode for seeding the Approved FIPS 186-2 PRNG.

fulfilling requests for large keys and thus ignoring other requests. 16384 bits is the recommended maximum strength with 8192 and 4096 still meeting the majority of current needs.

[4] Note: Per IG 7.5, only RSA keys of size 1024-15360 bit are acceptable for key transport. RSA keys of size below 1024 bit are non-compliant for key transport.

[5] Note: DH is non-compliant if less than 80-bits of encryption strength.

[6] Differs from the regular DSA and RSA functions in that data is not hashed before being signed.

		Available upon request from IBM. Patented by IBM, EC Pat. No. EP1081591A2, U.S. pat. Pend.

In addition, the module supports the following non-approved algorithms that must NOT be used in the FIPS-Approved mode:

Type	**Algorithm**	**Specification**
Symmetric Cipher	AES (GCM, CTS, and PCBC modes)	GCM, PCBC and CTS mode GCM (Cert. #2107 – Non-Compliant)
Symmetric Cipher	Triple-DES	PCBC and CTS mode
MAC	HMAC	Auth HMAC for SHA-256 truncated to 128 and for SHA-512 truncated to 256

7 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

8 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* Javadoc.

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* Javadoc.

8.1 Self Tests

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.

8.2 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.getInstance	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 Triple-DES, and AES algorithms with various cipher modes and paddings. MessageDigest allows for SHA-1, SHA-256, SHA-384, SHA-512, MD5 hashing.
Init Cipher.init	Intitializes the cryptographic object for use. This includes the mode (encryption or decryption)

MessageDigest.init	and the cryptographic key. This call is based on a cryptographic object.
Update Cipher.update MessageDigest.update	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

8.3 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.getInstance	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.

8.4 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.

8.5 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.

8.6 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).

8.7 KeyFactory

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

Service	Description
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.

9 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. The module does not provide authentication for any role.

All the services in the previous section are available to both the roles of the module. The role assumed by the operator is implicit based upon the service being invoked.

9.1 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. This role is also responsible for installing and removing the module.

9.2 Cryptographic User role

The User role has the responsibility for operating cryptographic functions on data. The available functions for the User role are listed in Appendix A of this document.

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.

10 Cryptographic Module Key Management

The module supports the use of the following cryptographic keys: Diffie-Hellman public/private keys, Triple-DES, AES, EC public/private keys, RSA public/private keys, DSA public/private keys, EC public/private keys, HMAC-SHA1, HMAC-SHA 256, HMAC-SHA 384, and HMAC-SHA 512.

CSP/Key Name	Key Type	Generation/Input	Output	Storage	Zeroization	Roles
AES key	symmetric	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
DSA Private Key	asymmetric	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
ECDSA Private Key	asymmetric	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
EC Diffie-Hellman Private Key	asymmetric	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
HMAC-SHA1 key	Mac	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
HMAC-SHA256 key	Mac	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User

HMAC-SHA384 key	Mac	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
HMAC-SHA512 key	Mac	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
RSA Private Key	asymmetric	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
Triple-DES key	symmetric	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
Integrity Key	Mac (HMAC SHA-1)	Hardcoded	Never	Harddisk	When the module is deleted	Crypto Officer
PRNG Seed Key	FIPS 186-2 RNG Seed Key	Generated internally using non-approved RNG	Never	RAM	When a new seed key is generated	Crypto Officer, Crypto User
PRNG Seed	FIPS 186-2 RNG Seed	Generated internally using non-approved RNG	Never	RAM	When a new seed is generated	Crypto Officer, Crypto User
DRBG InputString	SP 800-90A InputString	Passed from caller	Never	RAM	When new value is provided on reseed	Crypto Officer, Crypto User
DRBG Seed	SP 800-90A Seed	Passed from caller	Never	RAM	When reseed is called	Crypto Officer, Crypto User

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* Javadoc.

10.1 Key Generation

Symmetric keys are generated using the FIPS Approved FIPS 186-2 (Appendix 3.1 and 3.3) 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. The module also implements ECDSA key generation which is 186-2 and 186-3 compliant.

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 algorithm, if a seed value is not supplied to the PRNG by the user.

Please note that the strength of the generated keys is directly dependent on the "randomness" of the entropy used in the key generation process. Thus, if the random data collected does not have sufficient entropy, the strength of the generated keys could be modified.

10.2 Key Storage

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

10.3 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.

10.4 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

11 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-SHA1 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.

Power-up self-tests include known answer tests for the RSA, Diffie-Hellman, SHA-1, SHA-256, SHA-384, SHA-512, Triple-DES, AES, AES/GCM, DSA, ECDSA, HMAC SHA1, HMAC SHA256, HMAC SHA384, HMAC SHA512 cryptographic algorithms and FIPS 186-2 RNG, as well as SP 800-90A Hash-based DRBG. Should any self-test fail, the module transitions to the Error state and the self-test failure is reported back through the return values and error codes (FIPSRuntimeException) from the API call.

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

Additionally, conditional tests are performed when asymmetric keys are generated and random number generators are invoked. These tests include a continuous random number generator tests for the FIPS-approved PRNG, the FIPS-approved DRBG, and the non-FIPS-approved TRNG (used to seed FIPS 186-2 PRNG) as well as pair-wise consistency tests of the generated DSA, ECDSA and RSA keys.

12 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 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 or FIPSPRNG) and using that object whenever possible.
- MD5, RSAforSSL and DSAforSSL can only be used if the user is implementing the TLS protocol for Secure Sockets. Any other use will cause the application to be in non-compliance.

13 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 (in java.security) and allowing the JVM to select the first provider that has the requested cryptographic capability.

The module operator is required to provide the entropy source when initializing the Hash DRBG of the IBMJCE module. This entropy source will be used by the DRBG to obtain seeding information.

14 Cryptographic Module Operating system environment

14.1 Framework

The cryptographic module is dependent 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 6.0 or higher JVM must be available.

A valid commercial grade installation of a Java SDK 6.0 or higher JVM that includes the Java Cryptographic Extension framework (Version 6.0) 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 6.0/7.0 framework (herein referred to as "JCE FW"), and to assure authorized providers of the integrity and authenticity of the JCE FW that they plug into, JCE FW and its providers will engage in mutual authentication. Only providers that have authenticated JCE FW, and who in turn have been authenticated by JCE FW, will become usable in the JCE FW 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 framework is digitally signed. Providers that provide implementations for JCE FW 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 FW 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.

14.2 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 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 a 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 can be done by disabling any service (within the Administrative tools) that provider remote access (e.g. – ftp, telnet, ssh, and server) and disallowing multiple operators to log in at once.

14.3 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.

14.4 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.

15 Mitigation of other attacks

The IBMJCEFIPS provider has been obfuscated. The commercial product KlassMaster provides code obfuscation. This level of optimized code makes it difficult to decompile and reuse the derived source code. IBM's tests 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

RSA Blinding has been added to the RSA Signing and RSA encryption function to help mitigate timing attacks.

No other mitigation of other attacks is provided.

16 References

[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

17 Appendix A: Function List

The following is a list of the public functions found in this module. Please refer to the *IBM Java JCE FIPS (IBMJCEFIPS) Cryptographic Module API* document.

17.1 A

add(int, byte[]...) - Static method in class com.ibm.crypto.fips.provider.ByteAdder
Add a variable number of byte arrays of variable size.
add(int, byte[]...) - Static method in class com.ibm.crypto.fips.provider.ByteAdder
Add a variable number of byte arrays of variable size.
AESCipher - Class in com.ibm.crypto.fips.provider
This class implements the AES algorithm in its various modes (ECB, CFB, OFB, CBC, PCBC) and padding schemes (PKCS5Padding, NoPadding).
AESCipher - Class in com.ibm.crypto.fips.provider
This class implements the AES algorithm in its various modes (ECB, CFB, OFB, CBC, PCBC) and padding schemes (PKCS5Padding, NoPadding).
AESCipher() - Constructor for class com.ibm.crypto.fips.provider.AESCipher
Creates an instance of AES cipher with default ECB mode and PKCS5Padding.
AESCipher(String, String) - Constructor for class com.ibm.crypto.fips.provider.AESCipher
Creates an instance of AES cipher with the requested mode and padding.
AESCipher() - Constructor for class com.ibm.crypto.fips.provider.AESCipher

Creates an instance of AES cipher with default ECB mode and PKCS5Padding.

AESCipher(String, String) - Constructor for class com.ibm.crypto.fips.provider.AESCipher

Creates an instance of AES cipher with the requested mode and padding.

AESGCMCipher - Class in com.ibm.crypto.fips.provider

This is an implementation of AES GCM mode, which does not currently support Cipher.update.

AESGCMCipher - Class in com.ibm.crypto.fips.provider

This is an implementation of AES GCM mode, which does not currently support Cipher.update.

AESGCMCipher() - Constructor for class com.ibm.crypto.fips.provider.AESGCMCipher

Constructs a new AESGCMCipher instance.

AESGCMCipher() - Constructor for class com.ibm.crypto.fips.provider.AESGCMCipher

Constructs a new AESGCMCipher instance.

AESGCMCrypt - Class in com.ibm.crypto.fips.provider

Implementation of AESGCMCrypt, extending new SymmetricAEADCipher.

AESGCMCrypt - Class in com.ibm.crypto.fips.provider

Implementation of AESGCMCrypt, extending new SymmetricAEADCipher.

AESGCMCrypt() - Constructor for class com.ibm.crypto.fips.provider.AESGCMCrypt

AESGCMCrypt() - Constructor for class com.ibm.crypto.fips.provider.AESGCMCrypt

AESKeyFactory - Class in com.ibm.crypto.fips.provider

This class implements the AES key factory of the IBMJCEFIPS provider.

AESKeyFactory - Class in com.ibm.crypto.fips.provider

This class implements the AES key factory of the IBMJCEFIPS provider.

AESKeyFactory() - Constructor for class com.ibm.crypto.fips.provider.AESKeyFactory

Verify the JCE framework in the constructor.

AESKeyFactory() - Constructor for class com.ibm.crypto.fips.provider.AESKeyFactory

Verify the JCE framework in the constructor.

AESKeyGenerator - Class in com.ibm.crypto.fips.provider

This class generates a secret key for use with the AES algorithm.

AESKeyGenerator - Class in com.ibm.crypto.fips.provider

This class generates a secret key for use with the AES algorithm.

AESKeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.AESKeyGenerator

Verify the JCE framework in the constructor.

AESKeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.AESKeyGenerator

Verify the JCE framework in the constructor.

AESKeySpec - Class in com.ibm.crypto.fips.provider

This class specifies a AES key.

AESKeySpec - Class in com.ibm.crypto.fips.provider

This class specifies a AES key.

AESKeySpec(byte[]) - Constructor for class com.ibm.crypto.fips.provider.AESKeySpec

Uses the bytes in key as the key material for the AES key.

AESKeySpec(byte[], int, int) - Constructor for class com.ibm.crypto.fips.provider.AESKeySpec

Uses the bytes in `key`, beginning at `offset` inclusive, as the key material for the AES key.

AESKeySpec(byte[]) - Constructor for class com.ibm.crypto.fips.provider.AESKeySpec

Uses the bytes in `key` as the key material for the AES key.

AESKeySpec(byte[], int, int) - Constructor for class com.ibm.crypto.fips.provider.AESKeySpec

Uses the bytes in `key`, beginning at `offset` inclusive, as the key material for the AES key.

AESParameters - Class in com.ibm.crypto.fips.provider

This class implements the parameter (IV) used with the AES algorithm in feedback-mode.

AESParameters - Class in com.ibm.crypto.fips.provider

This class implements the parameter (IV) used with the AES algorithm in feedback-mode.

AESParameters() - Constructor for class com.ibm.crypto.fips.provider.AESParameters

AESParameters() - Constructor for class com.ibm.crypto.fips.provider.AESParameters

AESSecretKey - Class in com.ibm.crypto.fips.provider

This class represents a AES key.

AESSecretKey - Class in com.ibm.crypto.fips.provider

This class represents a AES key.

AESSecretKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.AESSecretKey

Create a AES key from a given key

AESSecretKey(byte[], int) - Constructor for class com.ibm.crypto.fips.provider.AESSecretKey

Uses the first 16, 20, or 24 bytes (T) in `key`, beginning at `offset`, as the AES key.

AESSecretKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.AESSecretKey

Create a AES key from a given key

AESSecretKey(byte[], int) - Constructor for class com.ibm.crypto.fips.provider.AESSecretKey

Uses the first 16, 20, or 24 bytes (T) in `key`, beginning at `offset`, as the AES key.

AlgorithmStatus - Interface in com.ibm.crypto.fips.provider

AlgorithmStatus - Interface in com.ibm.crypto.fips.provider

17.2B

ByteAdder - Class in com.ibm.crypto.fips.provider

Adds byte arrays, intended for unsigned/positive numbers only.

ByteAdder - Class in com.ibm.crypto.fips.provider

Adds byte arrays, intended for unsigned/positive numbers only.

callAEADConstructor() - Method in class com.ibm.crypto.fips.provider.GCMHelper

callAEADConstructor() - Method in class com.ibm.crypto.fips.provider.GCMHelper

callGCMConstructorIntBA(int, byte[]) - Method in class com.ibm.crypto.fips.provider.GCMHelper

callGCMConstructorIntBA(int, byte[]) - Method in class com.ibm.crypto.fips.provider.GCMHelper

callGetAAD(Object) - Method in class com.ibm.crypto.fips.provider.GCMHelper

callGetAAD(Object) - Method in class com.ibm.crypto.fips.provider.GCMHelper

callGetIV(Object) - Method in class com.ibm.crypto.fips.provider.GCMHelper

callGetIV(Object) - Method in class com.ibm.crypto.fips.provider.GCMHelper

callGetTLen(Object) - Method in class com.ibm.crypto.fips.provider.GCMHelper

callGetTLen(Object) - Method in class com.ibm.crypto.fips.provider.GCMHelper

callSetAAD(Object, byte[]) - Method in class com.ibm.crypto.fips.provider.GCMHelper

callSetAAD(Object, byte[]) - Method in class com.ibm.crypto.fips.provider.GCMHelper

checkKeyLengths(int, BigInteger, int, int) - Static method in class com.ibm.crypto.fips.provider.RSAKeyFactory
Check the length of an RSA key modulus/exponent to make sure it is not too short or long.

checkKeyLengths(int, BigInteger, int, int) - Static method in class com.ibm.crypto.fips.provider.RSAKeyFactory
Check the length of an RSA key modulus/exponent to make sure it is not too short or long.

CipherWithWrappingSpi - Class in com.ibm.crypto.fips.provider
This class entends the javax.crypto.CipherSpi class with a concrete implementation of the methods for wrapping and unwrapping keys.

CipherWithWrappingSpi - Class in com.ibm.crypto.fips.provider
This class entends the javax.crypto.CipherSpi class with a concrete implementation of the methods for wrapping and unwrapping keys.

CipherWithWrappingSpi() - Constructor for class com.ibm.crypto.fips.provider.CipherWithWrappingSpi

CipherWithWrappingSpi() - Constructor for class com.ibm.crypto.fips.provider.CipherWithWrappingSpi

clone() - Method in class com.ibm.crypto.fips.provider.HmacSHA1

clone() - Method in class com.ibm.crypto.fips.provider.HmacSHA1

clone() - Method in class com.ibm.crypto.fips.provider.HmacSHA256

clone() - Method in class com.ibm.crypto.fips.provider.HmacSHA256

clone() - Method in class com.ibm.crypto.fips.provider.HmacSHA384

clone() - Method in class com.ibm.crypto.fips.provider.HmacSHA384

clone() - Method in class com.ibm.crypto.fips.provider.HmacSHA512

clone() - Method in class com.ibm.crypto.fips.provider.HmacSHA512

clone() - Method in class com.ibm.crypto.fips.provider.SHA
Clones this object.
clone() - Method in class com.ibm.crypto.fips.provider.SHA
Clones this object.
clone() - Method in class com.ibm.crypto.fips.provider.SHA2
Clones this object.
clone() - Method in class com.ibm.crypto.fips.provider.SHA2
Clones this object.
clone() - Method in class com.ibm.crypto.fips.provider.SHA3
Clones this object.
clone() - Method in class com.ibm.crypto.fips.provider.SHA3
Clones this object.
clone() - Method in class com.ibm.crypto.fips.provider.SHA5
Clones this object.
clone() - Method in class com.ibm.crypto.fips.provider.SHA5
Clones this object.
com.ibm.crypto.fips - package com.ibm.crypto.fips

com.ibm.crypto.fips.provider - package com.ibm.crypto.fips.provider

com.ibm.crypto.fips.provider - package com.ibm.crypto.fips.provider

Copyright - Class in com.ibm.crypto.fips

Copyright() - Constructor for class com.ibm.crypto.fips.Copyright

17.4D

DatawithDSA - Class in com.ibm.crypto.fips.provider

DatawithDSA - Class in com.ibm.crypto.fips.provider

DatawithDSA() - Constructor for class com.ibm.crypto.fips.provider.DatawithDSA

Constructs a new instance of this class.

DatawithDSA() - Constructor for class com.ibm.crypto.fips.provider.DatawithDSA

Constructs a new instance of this class.

DatawithECDSA - Class in com.ibm.crypto.fips.provider

DatawithECDSA - Class in com.ibm.crypto.fips.provider

DatawithECDSA() - Constructor for class com.ibm.crypto.fips.provider.DatawithECDSA

DatawithECDSA() - Constructor for class com.ibm.crypto.fips.provider.DatawithECDSA

DatawithRSA - Class in com.ibm.crypto.fips.provider

This class implements signature without this algorithm doing the hashing with RSA

DatawithRSA - Class in com.ibm.crypto.fips.provider

This class implements signature without this algorithm doing the hashing with RSA

DatawithRSA() - Constructor for class com.ibm.crypto.fips.provider.DatawithRSA

Construct a blank RSA object.

DatawithRSA() - Constructor for class com.ibm.crypto.fips.provider.DatawithRSA

Construct a blank RSA object.

DEFAULT_DIGEST_ALG - Static variable in class com.ibm.crypto.fips.provider.HASHDRBG

DEFAULT_DIGEST_ALG - Static variable in class com.ibm.crypto.fips.provider.HASHDRBG

DEFAULT_STRENGTH - Static variable in class com.ibm.crypto.fips.provider.HASHDRBG

DEFAULT_STRENGTH - Static variable in class com.ibm.crypto.fips.provider.HASHDRBG

DEFAULT_TAG_LENGTH - Static variable in interface com.ibm.crypto.fips.provider.GCMConstants

DEFAULT_TAG_LENGTH - Static variable in interface com.ibm.crypto.fips.provider.GCMConstants

DESedeCipher - Class in com.ibm.crypto.fips.provider

This class implements the triple-DES algorithm (DES-EDE) in its various modes (ECB, CFB, OFB, CBC, PCBC) and padding schemes (PKCS5Padding, NoPadding).

DESedeCipher - Class in com.ibm.crypto.fips.provider

This class implements the triple-DES algorithm (DES-EDE) in its various modes (ECB, CFB, OFB, CBC, PCBC) and padding schemes (PKCS5Padding, NoPadding).

DESedeCipher() - Constructor for class com.ibm.crypto.fips.provider.DESedeCipher

Creates an instance of DESede cipher with default ECB mode and PKCS5Padding.

DESedeCipher(String, String) - Constructor for class com.ibm.crypto.fips.provider.DESedeCipher

Creates an instance of DESede cipher with the requested mode and padding.

DESedeCipher() - Constructor for class com.ibm.crypto.fips.provider.DESedeCipher

Creates an instance of DESede cipher with default ECB mode and PKCS5Padding.

DESedeCipher(String, String) - Constructor for class com.ibm.crypto.fips.provider.DESedeCipher

Creates an instance of DESede cipher with the requested mode and padding.

DESedeKey - Class in com.ibm.crypto.fips.provider

This class represents a DES-EDE key.

DESedeKey - Class in com.ibm.crypto.fips.provider

This class represents a DES-EDE key.

DESedeKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.DESedeKey

Creates a DES-EDE key from a given key.

DESedeKey(byte[], int) - Constructor for class com.ibm.crypto.fips.provider.DESedeKey

Uses the first 24 bytes in `key`, beginning at `offset`, as the DES-EDE key

DESedeKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.DESedeKey

Creates a DES-EDE key from a given key.

DESedeKey(byte[], int) - Constructor for class com.ibm.crypto.fips.provider.DESedeKey

Uses the first 24 bytes in `key`, beginning at `offset`, as the DES-EDE key

DESedeKeyFactory - Class in com.ibm.crypto.fips.provider

This class implements the DES-EDE key factory of the IBMJCEFIPS provider.

DESedeKeyFactory - Class in com.ibm.crypto.fips.provider

This class implements the DES-EDE key factory of the IBMJCEFIPS provider.

DESedeKeyFactory() - Constructor for class com.ibm.crypto.fips.provider.DESedeKeyFactory

Verify the JCE framework in the constructor.

DESedeKeyFactory() - Constructor for class com.ibm.crypto.fips.provider.DESedeKeyFactory

Verify the JCE framework in the constructor.

DESedeKeyGenerator - Class in com.ibm.crypto.fips.provider

This class generates a Triple DES key.

DESedeKeyGenerator - Class in com.ibm.crypto.fips.provider

This class generates a Triple DES key.

DESedeKeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.DESedeKeyGenerator

Verify the JCE framework in the constructor.

DESedeKeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.DESedeKeyGenerator

Verify the JCE framework in the constructor.

DESedeParameters - Class in com.ibm.crypto.fips.provider

This class implements the parameter (IV) used with the Triple DES algorithm in feedback-mode.

DESedeParameters - Class in com.ibm.crypto.fips.provider

This class implements the parameter (IV) used with the Triple DES algorithm in feedback-mode.

DESedeParameters() - Constructor for class com.ibm.crypto.fips.provider.DESedeParameters

DESedeParameters() - Constructor for class com.ibm.crypto.fips.provider.DESedeParameters

DHKeyAgreement - Class in com.ibm.crypto.fips.provider

This class implements the Diffie-Hellman key agreement protocol between any number of parties.

DHKeyAgreement - Class in com.ibm.crypto.fips.provider

This class implements the Diffie-Hellman key agreement protocol between any number of parties.

DHKeyAgreement() - Constructor for class com.ibm.crypto.fips.provider.DHKeyAgreement

Verify the JCE framework in the constructor.

DHKeyAgreement() - Constructor for class com.ibm.crypto.fips.provider.DHKeyAgreement

Verify the JCE framework in the constructor.

DHKeyFactory - Class in com.ibm.crypto.fips.provider

This class implements the Diffie-Hellman key factory of the IBMJCEFIPS provider.

DHKeyFactory - Class in com.ibm.crypto.fips.provider

This class implements the Diffie-Hellman key factory of the IBMJCEFIPS provider.

DHKeyFactory() - Constructor for class com.ibm.crypto.fips.provider.DHKeyFactory

Verify the JCE framework in the constructor.

DHKeyFactory() - Constructor for class com.ibm.crypto.fips.provider.DHKeyFactory

Verify the JCE framework in the constructor.

DHKeyPairGenerator - Class in com.ibm.crypto.fips.provider

This class represents the key pair generator for Diffie-Hellman key pairs.

DHKeyPairGenerator - Class in com.ibm.crypto.fips.provider

This class represents the key pair generator for Diffie-Hellman key pairs.

DHKeyPairGenerator() - Constructor for class com.ibm.crypto.fips.provider.DHKeyPairGenerator

DHKeyPairGenerator() - Constructor for class com.ibm.crypto.fips.provider.DHKeyPairGenerator

DHParameterGenerator - Class in com.ibm.crypto.fips.provider

DHParameterGenerator - Class in com.ibm.crypto.fips.provider

DHParameterGenerator() - Constructor for class com.ibm.crypto.fips.provider.DHParameterGenerator

DHParameterGenerator() - Constructor for class com.ibm.crypto.fips.provider.DHParameterGenerator

DHParameters - Class in com.ibm.crypto.fips.provider

This class implements the parameter set used by the Diffie-Hellman key agreement as defined in the PKCS #3 standard.

DHParameters - Class in com.ibm.crypto.fips.provider

This class implements the parameter set used by the Diffie-Hellman key agreement as defined in the PKCS #3 standard.

DHParameters() - Constructor for class com.ibm.crypto.fips.provider.DHParameters

DHParameters() - Constructor for class com.ibm.crypto.fips.provider.DHParameters

DHPrivateKey - Class in com.ibm.crypto.fips.provider

A private key in PKCS#8 format for the Diffie-Hellman key agreement algorithm.

DHPrivateKey - Class in com.ibm.crypto.fips.provider

A private key in PKCS#8 format for the Diffie-Hellman key agreement algorithm.

DHPrivateKey(BigInteger, BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.DHPrivateKey

Make a DH private key out of a private value x, a prime modulus p, and a base generator g.

DHPrivateKey(BigInteger, BigInteger, BigInteger, int) - Constructor for class com.ibm.crypto.fips.provider.DHPrivateKey

Make a DH private key out of a private value x, a prime modulus p, a base generator g, and a private-value length l.

DHPrivateKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.DHPrivateKey

Make a DH private key from its DER encoding (PKCS #8).

DHPrivateKey(BigInteger, BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.DHPrivateKey

Make a DH private key out of a private value x, a prime modulus p, and a base generator g.

DHPrivateKey(BigInteger, BigInteger, BigInteger, int) - Constructor for class com.ibm.crypto.fips.provider.DHPrivateKey

Make a DH private key out of a private value x, a prime modulus p, a base generator g, and a private-value length l.

DHPrivateKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.DHPrivateKey

Make a DH private key from its DER encoding (PKCS #8).

DHPublicKey - Class in com.ibm.crypto.fips.provider

A public key in X.509 format for the Diffie-Hellman key agreement algorithm.

DHPublicKey - Class in com.ibm.crypto.fips.provider

A public key in X.509 format for the Diffie-Hellman key agreement algorithm.

DHPublicKey(BigInteger, BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.DHPublicKey

Make a DH public key out of a public value y, a prime modulus p, and a base generator g.

DHPublicKey(BigInteger, BigInteger, BigInteger, int) - Constructor for class com.ibm.crypto.fips.provider.DHPublicKey

Make a DH public key out of a public value y, a prime modulus p, a base generator g, and a private-value length l.

DSAParameters() - Constructor for class com.ibm.crypto.fips.provider.DSAParameters

DSAPrivateKey - Class in com.ibm.crypto.fips.provider
This class represents an X.509 private key for the DSA Algorithm.
DSAPrivateKey - Class in com.ibm.crypto.fips.provider
This class represents an X.509 private key for the DSA Algorithm.
DSAPrivateKey(BigInteger, BigInteger, BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.DSAPrivateKey
Create a DSA private key from x, p, q, and g.
DSAPrivateKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.DSAPrivateKey
Create a DSA private key from it's DER encoding (PKCS#8)
DSAPrivateKey(BigInteger, BigInteger, BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.DSAPrivateKey
Create a DSA private key from x, p, q, and g.
DSAPrivateKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.DSAPrivateKey
Create a DSA private key from it's DER encoding (PKCS#8)
DSAPublicKey - Class in com.ibm.crypto.fips.provider
This class represents an X.509 public key for the DSA Algorithm.
DSAPublicKey - Class in com.ibm.crypto.fips.provider
This class represents an X.509 public key for the DSA Algorithm.
DSAPublicKey(BigInteger, BigInteger, BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.DSAPublicKey
Create a new DSA public key from y, p, q, and g.
DSAPublicKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.DSAPublicKey
Make a DSA public key from its DER encoding (X.509).
DSAPublicKey(BigInteger, BigInteger, BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.DSAPublicKey
Create a new DSA public key from y, p, q, and g.
DSAPublicKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.DSAPublicKey
Make a DSA public key from its DER encoding (X.509).

17.5E

EC_SIZE_192 - Static variable in class com.ibm.crypto.fips.provider.ECUtils

EC_SIZE_192 - Static variable in class com.ibm.crypto.fips.provider.ECUtils

EC_SIZE_224 - Static variable in class com.ibm.crypto.fips.provider.ECUtils

EC_SIZE_224 - Static variable in class com.ibm.crypto.fips.provider.ECUtils

EC_SIZE_256 - Static variable in class com.ibm.crypto.fips.provider.ECUtils

EC_SIZE_256 - Static variable in class com.ibm.crypto.fips.provider.ECUtils

Key implementation for EC private keys.
ECPrivateKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.ECPrivateKey
Construct a key from its encoding.
ECPrivateKey(BigInteger, ECParameterSpec) - Constructor for class com.ibm.crypto.fips.provider.ECPrivateKey
Construct a key from its components.
ECPrivateKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.ECPrivateKey
Construct a key from its encoding.
ECPrivateKey(BigInteger, ECParameterSpec) - Constructor for class com.ibm.crypto.fips.provider.ECPrivateKey
Construct a key from its components.
ECPublicKey - Class in com.ibm.crypto.fips.provider

ECPublicKey - Class in com.ibm.crypto.fips.provider

ECPublicKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.ECPublicKey

ECPublicKey(ECPoint, ECParameterSpec) - Constructor for class com.ibm.crypto.fips.provider.ECPublicKey

ECPublicKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.ECPublicKey

ECPublicKey(ECPoint, ECParameterSpec) - Constructor for class com.ibm.crypto.fips.provider.ECPublicKey

ECUtils - Class in com.ibm.crypto.fips.provider

ECUtils - Class in com.ibm.crypto.fips.provider

ECUtils() - Constructor for class com.ibm.crypto.fips.provider.ECUtils

ECUtils() - Constructor for class com.ibm.crypto.fips.provider.ECUtils

engineGenerateSeed(int) - Method in class com.ibm.crypto.fips.provider.SecureRandom

engineGenerateSeed(int) - Method in class com.ibm.crypto.fips.provider.SecureRandom

engineNextBytes(byte[]) - Method in class com.ibm.crypto.fips.provider.SecureRandom

engineNextBytes(byte[]) - Method in class com.ibm.crypto.fips.provider.SecureRandom

engineSetSeed(byte[]) - Method in class com.ibm.crypto.fips.provider.SecureRandom

17.6F

17.7G

Method returns the cryptographic modules FIPS 140-2 certification level
getFipsLevel() - Method in interface com.ibm.crypto.fips.provider.ModuleStatus
Method returns the cryptographic modules FIPS 140-2 certification level
getFipsLevel() - Method in interface com.ibm.crypto.fips.provider.ModuleStatus
Method returns the cryptographic modules FIPS 140-2 certification level
getFormat() - Method in class com.ibm.crypto.fips.provider.AESSecretKey

getFormat() - Method in class com.ibm.crypto.fips.provider.AESSecretKey

getFormat() - Method in class com.ibm.crypto.fips.provider.DESedeKey

getFormat() - Method in class com.ibm.crypto.fips.provider.DESedeKey

getFormat() - Method in class com.ibm.crypto.fips.provider.DHPrivateKey
Returns the encoding format of this key: "PKCS#8"
getFormat() - Method in class com.ibm.crypto.fips.provider.DHPrivateKey
Returns the encoding format of this key: "PKCS#8"
getFormat() - Method in class com.ibm.crypto.fips.provider.DHPublicKey
Returns the encoding format of this key: "X.509"
getFormat() - Method in class com.ibm.crypto.fips.provider.DHPublicKey
Returns the encoding format of this key: "X.509"
getGCMClass() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMClass() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMConstructorIntBA() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMConstructorIntBA() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMGetAAD() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMGetAAD() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMGetIV() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMGetIV() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMGetTLen() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMGetTLen() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMSetAAD() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getGCMSetAAD() - Method in class com.ibm.crypto.fips.provider.GCMHelper

getIV() - Method in interface com.ibm.crypto.fips.provider.FeedbackCipher
Gets the initialization vector.
getIV() - Method in interface com.ibm.crypto.fips.provider.FeedbackCipher
Gets the initialization vector.

getKey() - Method in class com.ibm.crypto.fips.provider.AESKeySpec
Returns the AES key material.
getKey() - Method in class com.ibm.crypto.fips.provider.AESKeySpec
Returns the AES key material.
getModulus() - Method in class com.ibm.crypto.fips.provider.RSAPrivateCrtKey
Return the modulus.
getModulus() - Method in class com.ibm.crypto.fips.provider.RSAPrivateCrtKey
Return the modulus.
getModulus() - Method in class com.ibm.crypto.fips.provider.RSAPrivateKey
Return the modulus.
getModulus() - Method in class com.ibm.crypto.fips.provider.RSAPrivateKey
Return the modulus.
getModulus() - Method in class com.ibm.crypto.fips.provider.RSAPublicKey
Return the modulus.
getModulus() - Method in class com.ibm.crypto.fips.provider.RSAPublicKey
Return the modulus.
getName() - Method in class com.ibm.crypto.fips.provider.ECNamedCurve

getName() - Method in class com.ibm.crypto.fips.provider.ECNamedCurve

getOIDFromName(String) - Static method in class com.ibm.crypto.fips.provider.ECNamedCurve
Accepts a name and attempts to retrieve the corresponding OID.
getOIDFromName(String) - Static method in class com.ibm.crypto.fips.provider.ECNamedCurve
Accepts a name and attempts to retrieve the corresponding OID.
getParams() - Method in class com.ibm.crypto.fips.provider.DHPrivateKey
Returns the key parameters.
getParams() - Method in class com.ibm.crypto.fips.provider.DHPrivateKey
Returns the key parameters.
getParams() - Method in class com.ibm.crypto.fips.provider.DHPublicKey
Returns the key parameters.
getParams() - Method in class com.ibm.crypto.fips.provider.DHPublicKey
Returns the key parameters.
getParams() - Method in class com.ibm.crypto.fips.provider.DSAPrivateKey
Returns the DSA parameters associated with this key, or null if the parameters could not be parsed.
getParams() - Method in class com.ibm.crypto.fips.provider.DSAPrivateKey
Returns the DSA parameters associated with this key, or null if the parameters could not be parsed.
getParams() - Method in class com.ibm.crypto.fips.provider.DSAPublicKey
Return the DSA parameters for the receiver.
getParams() - Method in class com.ibm.crypto.fips.provider.DSAPublicKey
Return the DSA parameters for the receiver.
getParams() - Method in class com.ibm.crypto.fips.provider.ECPrivateKey

getParams() - Method in class com.ibm.crypto.fips.provider.ECPrivateKey

getParams() - Method in class com.ibm.crypto.fips.provider.ECPublicKey

17.8H

Calculates a hash code value for the object.

hashCode() - Method in class com.ibm.crypto.fips.provider.DHPublicKey

Calculates a hash code value for the object.

HASHDRBG - Class in com.ibm.crypto.fips.provider

This class implements the HASH_DRBG algorithm found in NIST SP 800-90.

HASHDRBG - Class in com.ibm.crypto.fips.provider

This class implements the HASH_DRBG algorithm found in NIST SP 800-90.

HASHDRBG() - Constructor for class com.ibm.crypto.fips.provider.HASHDRBG

Construct a hash-based deterministic random bit generator with the appropriate algorithm for this amount of strength.

HASHDRBG() - Constructor for class com.ibm.crypto.fips.provider.HASHDRBG

Construct a hash-based deterministic random bit generator with the appropriate algorithm for this amount of strength.

HmacSHA1 - Class in com.ibm.crypto.fips.provider

This is an implementation of the HMAC-SHA1 algorithm.

HmacSHA1 - Class in com.ibm.crypto.fips.provider

This is an implementation of the HMAC-SHA1 algorithm.

HmacSHA1() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA1

Standard constructor, creates a new HmacSHA1 instance.

HmacSHA1() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA1

Standard constructor, creates a new HmacSHA1 instance.

HmacSHA1KeyGenerator - Class in com.ibm.crypto.fips.provider

This class generates a secret key for use with the HMAC-SHA1 algorithm.

HmacSHA1KeyGenerator - Class in com.ibm.crypto.fips.provider

This class generates a secret key for use with the HMAC-SHA1 algorithm.

HmacSHA1KeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA1KeyGenerator

Verify the JCE framework in the constructor.

HmacSHA1KeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA1KeyGenerator

Verify the JCE framework in the constructor.

HmacSHA256 - Class in com.ibm.crypto.fips.provider

This is an implementation of the HMAC-SHA256 algorithm.

HmacSHA256 - Class in com.ibm.crypto.fips.provider

This is an implementation of the HMAC-SHA256 algorithm.

HmacSHA256() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA256

Standard constructor, creates a new HmacSHA256 instance.

HmacSHA256() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA256

Standard constructor, creates a new HmacSHA256 instance.

HmacSHA256KeyGenerator - Class in com.ibm.crypto.fips.provider

This class generates a secret key for use with the HMAC-SHA256 algorithm.

HmacSHA256KeyGenerator - Class in com.ibm.crypto.fips.provider

This class generates a secret key for use with the HMAC-SHA256 algorithm.

HmacSHA256KeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA256KeyGenerator

Verify the JCE framework in the constructor.

HmacSHA256KeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA256KeyGenerator

Verify the JCE framework in the constructor.

HmacSHA384 - Class in com.ibm.crypto.fips.provider
This is an implementation of the HMAC-SHA384 algorithm.
HmacSHA384 - Class in com.ibm.crypto.fips.provider
This is an implementation of the HMAC-SHA384 algorithm.
HmacSHA384() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA384
Standard constructor, creates a new HmacSHA384 instance.
HmacSHA384() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA384
Standard constructor, creates a new HmacSHA384 instance.
HmacSHA384KeyGenerator - Class in com.ibm.crypto.fips.provider
This class generates a secret key for use with the HMAC-SHA384 algorithm.
HmacSHA384KeyGenerator - Class in com.ibm.crypto.fips.provider
This class generates a secret key for use with the HMAC-SHA384 algorithm.
HmacSHA384KeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA384KeyGenerator
Verify the JCE framework in the constructor.
HmacSHA384KeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA384KeyGenerator
Verify the JCE framework in the constructor.
HmacSHA512 - Class in com.ibm.crypto.fips.provider
This is an implementation of the HMAC-SHA512 algorithm.
HmacSHA512 - Class in com.ibm.crypto.fips.provider
This is an implementation of the HMAC-SHA512 algorithm.
HmacSHA512() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA512
Standard constructor, creates a new HmacSHA512 instance.
HmacSHA512() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA512
Standard constructor, creates a new HmacSHA512 instance.
HmacSHA512KeyGenerator - Class in com.ibm.crypto.fips.provider
This class generates a secret key for use with the HMAC-SHA512 algorithm.
HmacSHA512KeyGenerator - Class in com.ibm.crypto.fips.provider
This class generates a secret key for use with the HMAC-SHA512 algorithm.
HmacSHA512KeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA512KeyGenerator
Verify the JCE framework in the constructor.
HmacSHA512KeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.HmacSHA512KeyGenerator
Verify the JCE framework in the constructor.

17.9I

IBMJCEFIPS - Class in com.ibm.crypto.fips.provider
Defines the "IBMJCEFIPS" provider.
IBMJCEFIPS - Class in com.ibm.crypto.fips.provider
Defines the "IBMJCEFIPS" provider.
IBMJCEFIPS() - Constructor for class com.ibm.crypto.fips.provider.IBMJCEFIPS

IBMJCEFIPS() - Constructor for class com.ibm.crypto.fips.provider.IBMJCEFIPS

IBMOAEPParameters - Class in com.ibm.crypto.fips.provider

Initializes this key pair generator for the specified parameter set and source of randomness.

initialize(int, SecureRandom) - Method in class com.ibm.crypto.fips.provider.DHKeyPairGenerator

Initializes this key pair generator for a certain keysize and source of randomness.

initialize(AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.DHKeyPairGenerator

Initializes this key pair generator for the specified parameter set and source of randomness.

initialize(AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.DSAKeyPairGenerator

Initialize the receiver to use a given secure random generator, and generate keys from the provided set of parameters.

initialize(int, SecureRandom) - Method in class com.ibm.crypto.fips.provider.DSAKeyPairGenerator

Initialize the receiver to use a given secure random generator, and generate keys of a certain size.

initialize(AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.DSAKeyPairGenerator

Initialize the receiver to use a given secure random generator, and generate keys from the provided set of parameters.

initialize(int, SecureRandom) - Method in class com.ibm.crypto.fips.provider.DSAKeyPairGenerator

Initialize the receiver to use a given secure random generator, and generate keys of a certain size.

initialize(int, SecureRandom) - Method in class com.ibm.crypto.fips.provider.ECKeyPairGenerator

initialize(AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.ECKeyPairGenerator

initialize(int, SecureRandom) - Method in class com.ibm.crypto.fips.provider.ECKeyPairGenerator

initialize(AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.ECKeyPairGenerator

initialize(AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.RSAKeyPairGenerator

initialize(int, SecureRandom) - Method in class com.ibm.crypto.fips.provider.RSAKeyPairGenerator

Initializes this KeyPairGenerator for given modulus and random source

initialize(int) - Method in class com.ibm.crypto.fips.provider.RSAKeyPairGenerator

initialize(AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.RSAKeyPairGenerator

internalInit() - Method in class com.ibm.crypto.fips.provider.SHA3
Initialize the SHA3 information
internalInit() - Method in class com.ibm.crypto.fips.provider.SHA3
Initialize the SHA3 information
internalInit() - Method in class com.ibm.crypto.fips.provider.SHA5
Initialize the SHA5 information
internalInit() - Method in class com.ibm.crypto.fips.provider.SHA5
Initialize the SHA5 information
internalToString() - Method in class com.ibm.crypto.fips.provider.DatawithDSA
Answers a string containing a concise, human-readable description of the receiver.
internalToString() - Method in class com.ibm.crypto.fips.provider.DatawithDSA
Answers a string containing a concise, human-readable description of the receiver.
internalToString() - Method in class com.ibm.crypto.fips.provider.SHA1withDSA

internalToString() - Method in class com.ibm.crypto.fips.provider.SHA1withDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESCipher

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESCipher

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESGCMCipher

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESGCMCipher

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESKeyFactory

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESKeyFactory

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESKeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESKeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESKeySpec

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESKeySpec

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESParameters

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.AESParameters

isFipsApproved() - Method in interface com.ibm.crypto.fips.provider.AlgorithmStatus
Module identifies if the cryptographic operation (algorithm) is FIPS certified
isFipsApproved() - Method in interface com.ibm.crypto.fips.provider.AlgorithmStatus
Module identifies if the cryptographic operation (algorithm) is FIPS certified
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.CipherWithWrappingSpi

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.CipherWithWrappingSpi

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DatawithDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DatawithDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DatawithRSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DatawithRSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DESedeCipher

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DESedeCipher

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DESedeKeyFactory

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DESedeKeyFactory

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DESedeKeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DESedeKeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DESedeParameters

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DESedeParameters

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DHKeyAgreement
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DHKeyAgreement
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DHKeyFactory
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DHKeyFactory
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DHKeyPairGenerator
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DHKeyPairGenerator
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DHParameterGenerator
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DHParameterGenerator
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DHParameters

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DHParameters

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DSAKeyFactory

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DSAKeyFactory

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DSAKeyPairGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DSAKeyPairGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DSAParameterGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DSAParameterGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DSAParameters

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.DSAParameters

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.ECDHKeyAgreement
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.ECDHKeyAgreement
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.ECKeyFactory

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.ECKeyFactory

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.ECKeyPairGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.ECKeyPairGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.GCMParameterGenerator
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.GCMParameterGenerator
This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.GCMParameters

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.GCMParameters

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HASHDRBG

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HASHDRBG

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA1

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA1

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA1KeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA1KeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA256

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA256

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA256KeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA256KeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA384

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA384

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA384KeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA384KeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA512

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA512

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA512KeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.HmacSHA512KeyGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.RSA
This function allows an application to verify the the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.RSA
This function allows an application to verify the the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.RSAKeyFactory

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.RSAKeyFactory

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.RSAKeyPairGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.RSAKeyPairGenerator

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.RSASSL

This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.RSASSL

This function allows an application to verify the the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SecureRandom

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SecureRandom

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA1withDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA1withDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA1withECDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA1withECDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA1withRSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA1withRSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA2

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA2

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA2withECDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA2withECDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA2withRSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA2withRSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA3

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA3

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA3withECDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA3withECDSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA3withRSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA3withRSA

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA5

17.10 M

17.11 P

RSAPrivateCrtKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.RSAPrivateCrtKey
Make a RSA private key from its DER encoding (PKCS #8).
RSAPrivateCrtKey(BigInteger, BigInteger, BigInteger, BigInteger, BigInteger, BigInteger, BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.RSAPrivateCrtKey
This constructor computes missing key values and formats key values.
RSAPrivateCrtKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.RSAPrivateCrtKey
Make a RSA private key from its DER encoding (PKCS #8).
RSAPrivateKey - Class in com.ibm.crypto.fips.provider
An X.509 private key for the RSA Algorithm.
RSAPrivateKey - Class in com.ibm.crypto.fips.provider
An X.509 private key for the RSA Algorithm.
RSAPrivateKey(BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.RSAPrivateKey
Make a RSA private key.
RSAPrivateKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.RSAPrivateKey
Make a RSA private key from its DER encoding (PKCS #8).
RSAPrivateKey(BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.RSAPrivateKey
Make a RSA private key.
RSAPrivateKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.RSAPrivateKey
Make a RSA private key from its DER encoding (PKCS #8).
RSAPublicKey - Class in com.ibm.crypto.fips.provider
An X.509 public key for the RSA Algorithm.
RSAPublicKey - Class in com.ibm.crypto.fips.provider
An X.509 public key for the RSA Algorithm.
RSAPublicKey(BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.RSAPublicKey
Make a RSA public key.
RSAPublicKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.RSAPublicKey
Make a RSA public key from its DER encoding (X.509).
RSAPublicKey(BigInteger, BigInteger) - Constructor for class com.ibm.crypto.fips.provider.RSAPublicKey
Make a RSA public key.
RSAPublicKey(byte[]) - Constructor for class com.ibm.crypto.fips.provider.RSAPublicKey
Make a RSA public key from its DER encoding (X.509).
RSASSL - Class in com.ibm.crypto.fips.provider
This class uses the RSA class with blinding turned on.
RSASSL - Class in com.ibm.crypto.fips.provider
This class uses the RSA class with blinding turned on.
RSASSL() - Constructor for class com.ibm.crypto.fips.provider.RSASSL
Creates an instance of RSASSL
RSASSL() - Constructor for class com.ibm.crypto.fips.provider.RSASSL

Creates an instance of RSASSL

runSelfTest() - Method in class com.ibm.crypto.fips.provider.SelfTest

Method initiates a new self test

runSelfTest() - Method in class com.ibm.crypto.fips.provider.SelfTest

Method initiates a new self test

17.13 S

SecureRandom - Class in com.ibm.crypto.fips.provider

This class provides a cryptographically strong pseudo-random number generator based on the SHA1 message digest algorithm.

SecureRandom - Class in com.ibm.crypto.fips.provider

This class provides a cryptographically strong pseudo-random number generator based on the SHA1 message digest algorithm.

SecureRandom() - Constructor for class com.ibm.crypto.fips.provider.SecureRandom

SecureRandom(byte[]) - Constructor for class com.ibm.crypto.fips.provider.SecureRandom

SecureRandom() - Constructor for class com.ibm.crypto.fips.provider.SecureRandom

SecureRandom(byte[]) - Constructor for class com.ibm.crypto.fips.provider.SecureRandom

SelfTest - Class in com.ibm.crypto.fips.provider

SelfTest - Class in com.ibm.crypto.fips.provider

SelfTest() - Constructor for class com.ibm.crypto.fips.provider.SelfTest

SelfTest() - Constructor for class com.ibm.crypto.fips.provider.SelfTest

setT(int) - Method in class com.ibm.crypto.fips.provider.GCTR

Sets the length of the tag in bits.

setT(int) - Method in class com.ibm.crypto.fips.provider.GCTR

Sets the length of the tag in bits.

setupH(byte[]) - Method in class com.ibm.crypto.fips.provider.GhashMD

setupH(byte[]) - Method in class com.ibm.crypto.fips.provider.GhashMD

SHA - Class in com.ibm.crypto.fips.provider

This class implements the Secure Hash Algorithm (SHA) developed by the National Institute of Standards and Technology along with the National Security Agency.

SHA - Class in com.ibm.crypto.fips.provider

This class implements the Secure Hash Algorithm (SHA) developed by the National Institute of Standards and Technology along with the National Security Agency.

SHA() - Constructor for class com.ibm.crypto.fips.provider.SHA

Standard constructor, creates a new SHA instance, allocates its buffers from the heap.

SHA() - Constructor for class com.ibm.crypto.fips.provider.SHA

Standard constructor, creates a new SHA instance, allocates its buffers from the heap.

SHA1_MAX_STRENGTH - Static variable in class com.ibm.crypto.fips.provider.HASHDRBG

SHA1_MAX_STRENGTH - Static variable in class com.ibm.crypto.fips.provider.HASHDRBG

SHA1withDSA - Class in com.ibm.crypto.fips.provider

SHA1withDSA - Class in com.ibm.crypto.fips.provider

SHA1withDSA() - Constructor for class com.ibm.crypto.fips.provider.SHA1withDSA

Constructs a new instance of this class.

SHA1withDSA() - Constructor for class com.ibm.crypto.fips.provider.SHA1withDSA

Constructs a new instance of this class.

SHA1withECDSA - Class in com.ibm.crypto.fips.provider

SHA1withECDSA - Class in com.ibm.crypto.fips.provider

SHA1withECDSA() - Constructor for class com.ibm.crypto.fips.provider.SHA1withECDSA

SHA1withECDSA() - Constructor for class com.ibm.crypto.fips.provider.SHA1withECDSA

SHA1withRSA - Class in com.ibm.crypto.fips.provider

This class implements the SHA1withRSA

SHA1withRSA - Class in com.ibm.crypto.fips.provider

This class implements the SHA1withRSA

SHA1withRSA() - Constructor for class com.ibm.crypto.fips.provider.SHA1withRSA

Construct a blank RSA object.

SHA1withRSA() - Constructor for class com.ibm.crypto.fips.provider.SHA1withRSA

Construct a blank RSA object.

SHA2 - Class in com.ibm.crypto.fips.provider

This class implements the Secure Hash Algorithm 2 (SHA2) developed by the National Institute of Standards and Technology along with the National Security Agency.

SHA2 - Class in com.ibm.crypto.fips.provider

This class implements the Secure Hash Algorithm 2 (SHA2) developed by the National Institute of Standards and Technology along with the National Security Agency.

SHA2() - Constructor for class com.ibm.crypto.fips.provider.SHA2

Standard constructor, creates a new SHA2 instance, allocates its buffers from the heap.

SHA2() - Constructor for class com.ibm.crypto.fips.provider.SHA2

Standard constructor, creates a new SHA2 instance, allocates its buffers from the heap.

SHA224_MAX_STRENGTH - Static variable in class com.ibm.crypto.fips.provider.HASHDRBG

SHA224_MAX_STRENGTH - Static variable in class com.ibm.crypto.fips.provider.HASHDRBG

SHA256withDSA - Class in com.ibm.crypto.fips.provider

SHA256withDSA - Class in com.ibm.crypto.fips.provider

SHA256withDSA() - Constructor for class com.ibm.crypto.fips.provider.SHA256withDSA
Constructs a new instance of this class.

SHA256withDSA() - Constructor for class com.ibm.crypto.fips.provider.SHA256withDSA
Constructs a new instance of this class.

SHA2withECDSA - Class in com.ibm.crypto.fips.provider

SHA2withECDSA - Class in com.ibm.crypto.fips.provider

SHA2withECDSA() - Constructor for class com.ibm.crypto.fips.provider.SHA2withECDSA

SHA2withECDSA() - Constructor for class com.ibm.crypto.fips.provider.SHA2withECDSA

SHA2withRSA - Class in com.ibm.crypto.fips.provider
This class implements the SHA1withRSA

SHA2withRSA - Class in com.ibm.crypto.fips.provider
This class implements the SHA1withRSA

SHA2withRSA() - Constructor for class com.ibm.crypto.fips.provider.SHA2withRSA
Construct a blank RSA object.

SHA2withRSA() - Constructor for class com.ibm.crypto.fips.provider.SHA2withRSA
Construct a blank RSA object.

SHA3 - Class in com.ibm.crypto.fips.provider
This class implements the Secure Hash Algorithm 3 (SHA-3) developed by the National Institute of Standards and Technology along with the National Security Agency.

SHA3 - Class in com.ibm.crypto.fips.provider
This class implements the Secure Hash Algorithm 3 (SHA-3) developed by the National Institute of Standards and Technology along with the National Security Agency.

SHA3() - Constructor for class com.ibm.crypto.fips.provider.SHA3
Standard constructor, creates a new SHA3 instance, allocates its buffers from the heap.

SHA3() - Constructor for class com.ibm.crypto.fips.provider.SHA3
Standard constructor, creates a new SHA3 instance, allocates its buffers from the heap.

SHA3withECDSA - Class in com.ibm.crypto.fips.provider

17.14 T

17.15 U

17.16 Z

zeroize() - Method in class com.ibm.crypto.fips.provider.HASHDRBG
This function zeroizes the class so that nothing is in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.HASHDRBG
This function zeroizes the class so that nothing is in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.HmacSHA1
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.HmacSHA1
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.HmacSHA256
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.HmacSHA256
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.HmacSHA384
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.HmacSHA384
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.HmacSHA512
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.HmacSHA512
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.RSAPrivateCrtKey
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.RSAPrivateCrtKey
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.RSAPrivateKey
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.RSAPrivateKey
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.RSAPublicKey
This function zeroizes the key so that it isn't in memory.
zeroize() - Method in class com.ibm.crypto.fips.provider.RSAPublicKey
This function zeroizes the key so that it isn't in memory.

18 Notices

Java is a registered trademark of Oracle. Inc.

AIX, z/OS, AS/400 and IBM are trademarks or registered trademarks of IBM Corporation in the United States, other countries, or both.

HP-UX is a registered trademark Hewlet Packard, Inc

Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.