IBM Java JCE FIPS 140-2 Cryptographic Module

Security Policy

IBM JAVA JCE FIPS 140-2 Cryptographic Module
Revision: 1.5

July 2005

Status: Final

1.5 Edition (July 2005)
This edition applies to the 1.5 Edition of the IBMJCEFIPS – Security Policy and to all subsequent versions until otherwise indicated in new editions.

Table of Contents

Introduction

The IBM® Java® JCE (Java Cryptographic Extension) FIPS 140-2 Cryptographic Module (Version 1.2) 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 for Solaris®, Windows®, AIX®, z/OS®, AS/400®, Linux® (Red Hat and SuSE®)

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

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

Operation of the Cryptographic Module

The cryptographic module must be utilized in a 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.

The module provides for two operator roles:

- Crypto Officer
- User

There is no maintenance role in this cryptographic module.

An application must use the IBMJCEFIPS provider to enable the use of appropriate cryptographic functions in a FIPS approved manner. The application calling the IBMJCEFIPS provider must understand the roles of the APIs, Crypto Officer vs. User. The *Cryptographic Module Roles* section of this document details the APIs that apply to each role. 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.

Changes from Version 1.1 to 1.2

The following was added to the 1.2 version of IBMJCEFIPS:

- The following hash algorithms have been added: SHA-256, SHA-384 and SHA-512.
- The non-approved MD5 algorithm was added for use by applications that want to implement the TLS protocol.
- Signature algorithms named DSAforSSL and RSAforSSL that create and verify signatures but require the application to hash the data before it is passed into these algorithms. The RSAforSSL does RSA blinding to help protect RSA private keys. Both these signature algorithms are non-FIPS Approved.
- New Cipher algorithm called RSAforSSL which does RSA blinding to protect RSA private keys. This is equal to doing RSA\SSL\PKCS1Padding using the RSA cipher algorithm name. RSAforSSL can only be used for key wrapping/unwrapping in a FIPS mode.
- Added a SecureRandom alias called FIPSPRNG.
- DES algorithm was removed from the module considering that NIST would be phasing DES out soon.
- The TDCP class was added. This class creates a TDES cipher object with CBC mode and no padding using the underlying TDES cipher for actual encryption/decryption.

Cryptographic Module Specification

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

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

Overall	**Security Level 1**

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

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

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

The IBMJCEFIPS provider was tested on a machine running Microsoft Windows XP Professional operating system in single-user mode with JVM 1.4.2. The software module maintains compliance when running on the Microsoft Windows 95, ® Microsoft Windows 98®, Microsoft Windows Me®, Microsoft Windows NT®, Microsoft Windows 2000®, and Microsoft Windows XP® operating systems, as well as, JVMs at the 5.x level, 1.4.x level and 1.3.1 level on those operating systems.

Since, at FIPS 140-2 Security level 1 the validation is independent of the Operating System, this validation is also applicable to AIX, Solaris, HP, Red Hat Linux, SuSE® Linux, z/OS® and IBM Operating System/400.

The module supports the following approved algorithms:

Type	**Algorithm**	**Specification**
Symmetric Cipher	AES (ECB, CBC, OFB and CFB modes)	FIPS 197
	Triple DES (ECB, CBC, OFB and CFB modes)	FIPS 46-3
Message Digest	SHA1 SHA-256 SHA-384 SHA-512 HMAC–SHA-1	FIPS 180-2 FIPS 198a
Random Number Generation	FIPS 186-2 Appendix 3.1	FIPS 186-2

Digital Signature	DSA (512 – 1024)	FIPS 186-2
Digital Signature	RSA (1024 – 2048)	PKCS#1 v1.5
Random Number Generation	FIPS 186-2 (Appendix 3.1) (SHA-1 based)	FIPS 186-2

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

Type	**Algorithm**	**Specification**
Random Number Generation	Universal Software Based Random Number Generator	Available upon request from IBM. Patented by IBM, EC Pat. No. EP1081591A2, U.S. pat. Pend.
Message Digest	MD5	RCF 1321 (Allowed for use within the TLS protocol).
Asymmetric Cipher	RSA	PKCS #1 with and without blinding (Allowed in the Approved mode for key transport)
Key Agreement	Diffie-Hellman (256 –1024)	PKCS #3 (Allowed in Approved mode)
Digital Signature	DSAforSSL	Allowed for use within the TLS protocol
Digital Signature	RSAforSSL	Allowed for use within the TLS protocol

Cryptographic Module Interfaces

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

The cryptographic module is classified as a “multi-chip standalone module” for FIPS 140-2 purposes. Thus, the module’s physical interfaces consist of those found as part of the computer’s hardware, such as the keyboard, mouse, disk

drive, CD drive, network adapters, serial and USB ports, monitor, speakers, etc. The module's logical interface is provided through the documented API.

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

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

Cryptographic Module Services

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

Self Test

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

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

Data Encryption/Decryption and Hashing (Digest)

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

Services	Description
getInstance	Creates a cryptographic object (Cipher/MessageDigest) for a selected algorithm. Also used to select the

Cipher.getInstance MessageDigest.getInstance	cryptographic provider to be used by that object. Cipher allows for 3DES, 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 MessageDigest.init	Intitializes the cryptographic object for use. This includes the mode (encryption or decryption) and the cryptographic key. This call is based on a cryptographic object.
Update Cipher.update MessageDigest.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

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.

Key Security

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

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

Signature

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

Service	Description
getInstance Signature.getInstance	Creates a cryptographic object (Signature) for a selected algorithm. Also used to select the cryptographic provider to be used by that object.
InitSign	Intitializes the cryptographic object for use. This includes the cryptographic private key. This call is based on a cryptographic object.
Update	Update a byte or byte array in the data to be signed or verified. This call is based on a cryptographic object.

Sign	Get message digest for all the data thus far updated, then sign the message digest. This call is based on a cryptographic object.
InitVerify	Intitializes the cryptographic object for use. This includes the cryptographic public key. This call is based on a cryptographic object.
verify	Verify the signature (compare the result with the message digest). This call is based on a cryptographic object.

Secret Key Factory

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

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

KeyFactory

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

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

	provided key specification (key material).
getKeySpec	Returns a specification (key material) of the given key object in the requested format.

Cryptographic Module Roles

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

Cryptographic Officer role

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

Cryptographic User role

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

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

There is no maintenance role.

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

Cryptographic Module Key Management

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

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

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.

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.

Key Storage

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

Key Protection

The management and allocation of memory is the responsibility of the operating system. It is assumed that a unique process space is allocated for each request, and that the operating system and the underlying central processing unit (CPU) hardware control access to that space.

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

Key Zeroization

All cryptographic keys and contexts are zeroized when an operator:

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

Cryptographic Module Self-Tests

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

The integrity of the module is verified by performing a HMAC-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, SHA1, SHA-256, SHA-384, SHA-512, Triple DES, AES, DSA, HMAC SHA1 cryptographic algorithms and pseudo random number generation. Should any self-test fail, the module transitions to the Error state.

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 test and pair-wise consistency tests of the generated DSA and RSA keys.

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 when ever 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.

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.

Cryptographic Module Operating system environment

Framework

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

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

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

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

In addition, each provider does do self-integrity checking to ensure that the JAR file containing its code has not been tampered with. The JCE 1.2.1 framework is digitally signed. Providers that provide implementations for JCE 1.2.1 services must also be digitally signed. Authentication includes verification of those signatures and ensuring the signatures were generated by trusted entities.

Certain Certificate Authorities are deemed to be "trusted" and any code signed using a certificate that can be traced up a certificate chain to a certificate for one of the trusted Certificate Authorities are considered trusted. Both JCE 1.2.1 and provider packages do embed within themselves the bytes for the certificates for the relevant trusted Certificate Authorities. At runtime, the embedded certificates will be used in determining whether or not code is authentic. Currently, there are two trusted Certification Authorities: Sun Microsystems' JCE Code Signing CA, and IBM JCE Code Signing CA.

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

Single user access (operating system requirements)

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

Java object model

The use of Java objects within the cryptographic module. In Java each cryptographic object is unique. Thus when an application generates a

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

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

Operating system restriction

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

Mitigation of other attacks

The IBMJCEFIPS provider has been obfuscated. 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.

[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

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.

A

addIdentity(Identity) - Method in class com.ibm.crypto.fips.provider.**IdentityDatabase**
Adds an identity to the database.
AESCipher - class com.ibm.crypto.fips.provider.**AESCipher**.
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.
AESKeyFactory - class com.ibm.crypto.fips.provider.**AESKeyFactory**.
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.
AESKeyGenerator - class com.ibm.crypto.fips.provider.**AESKeyGenerator**.
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.
AESKeySpec - class com.ibm.crypto.fips.provider.**AESKeySpec**.
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.
AESParameters - class com.ibm.crypto.fips.provider.**AESParameters**.
This class implements the parameter (IV) used with the AES algorithm in feedback-mode.
AESParameters() - Constructor for class com.ibm.crypto.fips.provider.**AESParameters**
This is the constructor for this class.
AESSecretKey - class com.ibm.crypto.fips.provider.**AESSecretKey**.
This class represents an AES key.
AlgorithmStatus - interface com.ibm.crypto.fips.provider.**AlgorithmStatus**.
This class can be used to identify if the cryptographic operation (algorithm) is FIPS certified

---

B

BEGIN_CERT - Static variable in class com.ibm.crypto.fips.provider.**X509Factory**
String that identifies the beginning of a certificate.

C

CipherWithWrappingSpi - class com.ibm.crypto.fips.provider.**CipherWithWrappingSpi**.
This class extends 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**
This is the constructor for this class.
clone() - Method in class com.ibm.crypto.fips.provider.**SHA**
Clones this object.
clone()- Method in class com.ibm.crypto.fips.provider.**MD5**
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.**SHA5**
Clones this object.
clone() - Method in class com.ibm.crypto.fips.provider.**HmacSHA1**
Clones this object.
com.ibm.crypto.fips.provider - package com.ibm.crypto.fips.provider
The package for this cryptographic module.

D

DatawithDSA - class com.ibm.crypto.fips.provider**.DatawithDSA.**
DatawithDSA() - Constructor for class com.ibm.crypto.fips.provider.**DatawithDSA**
Constructs a new instance of this class.
DatawithRSA - class com.ibm.crypto.fips.provider**.DatawithRSA.**
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.
DESedeCipher - class com.ibm.crypto.fips.provider.**DESedeCipher**.
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.
DESedeKey - class com.ibm.crypto.fips.provider.**DESedeKey**.
This class represents a DES-EDE key.
DESedeKeyFactory - class com.ibm.crypto.fips.provider.**DESedeKeyFactory**.
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.
DESedeKeyGenerator - class com.ibm.crypto.fips.provider.**DESedeKeyGenerator**.
This class generates a Triple DES key.
DESedeKeyGenerator() - Constructor for class com.ibm.crypto.fips.provider.**DESedeKeyGenerator**
Verify the JCE framework in the constructor.
DESedeParameters - class com.ibm.crypto.fips.provider.**DESedeParameters**.
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**
This is the constructor for this class.
DHKeyAgreement - class com.ibm.crypto.fips.provider.**DHKeyAgreement**.

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.
DHKeyFactory - class com.ibm.crypto.fips.provider.**DHKeyFactory**.
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.
DHKeyPairGenerator - class com.ibm.crypto.fips.provider.**DHKeyPairGenerator**.
This class represents the key pair generator for Diffie-Hellman key pairs.
DHKeyPairGenerator() - Constructor for class com.ibm.crypto.fips.provider.**DHKeyPairGenerator**
This is the constructor for this class.
DHParameterGenerator - class com.ibm.crypto.fips.provider.**DHParameterGenerator**.
This class is used to generate DH parameters.
DHParameterGenerator() - Constructor for class com.ibm.crypto.fips.provider.**DHParameterGenerator**
This is the constructor for this class.
DHParameters - class com.ibm.crypto.fips.provider.**DHParameters**.
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**
This is the constructor for this class.
DHPrivateKey - class com.ibm.crypto.fips.provider.**DHPrivateKey**.
A private key in PKCS#8 format for the Diffie-Hellman key agreement algorithm.
DHPublicKey - class com.ibm.crypto.fips.provider.**DHPublicKey**.
A public key in X.509 format for the Diffie-Hellman key agreement algorithm.
DSAKeyFactory - class com.ibm.crypto.fips.provider.**DSAKeyFactory**.
This class is a concrete implementaion of key factory for DSA.
DSAKeyFactory() - Constructor for class com.ibm.crypto.fips.provider.**DSAKeyFactory**
Constructs a new instance of this class.
DSAKeyPairGenerator - class com.ibm.crypto.fips.provider.**DSAKeyPairGenerator**.
This class is a concrete implementation for the generation of a pair of DSA keys
DSAKeyPairGenerator() - Constructor for class com.ibm.crypto.fips.provider.**DSAKeyPairGenerator**
This is the constructor for this class.
DSAParameterGenerator - class com.ibm.crypto.fips.provider.**DSAParameterGenerator**.

This class generates parameters for the DSA signature.
DSAParameterGenerator() - Constructor for class com.ibm.crypto.fips.provider.**DSAParameterGenerator**
Constructs a new instance of this class.
DSAParameters - class com.ibm.crypto.fips.provider.**DSAParameters**.
This class implements Digital Signature Algorithm paremters specified by com.ibm.crypto.fips.provider 186 standard.
DSAParameters() - Constructor for class com.ibm.crypto.fips.provider.**DSAParameters**
This is the constructor for this class.
DSAPrivateKey - class com.ibm.crypto.fips.provider.**DSAPrivateKey**.
This class represents an X.509 private key for the DSA Algorithm.
DSAPublicKey - class com.ibm.crypto.fips.provider.**DSAPublicKey**.
This class represents an X.509 public key for the DSA Algorithm.

E

END_CERT - Static variable in class com.ibm.crypto.fips.provider.**X509Factory**

engineGenerateCertificate(InputStream) - Method in class com.ibm.crypto.fips.provider.**X509Factory**
Generates an X.509 certificate object and initializes it with the data read from the input stream is.
engineGenerateCertificates(InputStream) - Method in class com.ibm.crypto.fips.provider.**X509Factory**
Returns a (possibly empty) collection view of X.509 certificates read from the given input stream is.
engineGenerateCertPath(InputStream) - Method in class com.ibm.crypto.fips.provider.**X509Factory**
Generates a CertPath object and initializes it with the data read from the input stream inStream.
engineGenerateCertPath(InputStream, String) - Method in class com.ibm.crypto.fips.provider.**X509Factory**
Generates a CertPath object and initializes it with the data read from the input stream inStream.
engineGenerateCertPath(List) - Method in class com.ibm.crypto.fips.provider.**X509Factory**
Generates a CertPath object and initializes it with the list of certificates supplied.
engineGenerateCRL(InputStream) - Method in class com.ibm.crypto.fips.provider.**X509Factory**
Generates an X.509 certificate revocation list (CRL) object and initializes it with the data read from the given input stream is.
engineGenerateCRLs(InputStream) - Method in class com.ibm.crypto.fips.provider.**X509Factory**

Returns a (possibly empty) collection view of X.509 CRLs read from the given input stream is.
engineGenerateSeed(int) - Method in class com.ibm.crypto.fips.provider.**SecureRandom**
Generates a seed of the length passed in.
engineGetCertPathEncodings() - Method in class com.ibm.crypto.fips.provider.**X509Factory**
Returns the encodings supported by this certification path factory, with the default encoding first.
engineNextBytes(byte[]) - Method in class com.ibm.crypto.fips.provider.**SecureRandom**
Generates random data of the length of the array passed in.
engineSetSeed(byte[]) - Method in class com.ibm.crypto.fips.provider.**SecureRandom**
Sets the set based on the byte array passed in.
equals(Object) - Method in class com.ibm.crypto.fips.provider.**DESedeKey**
Determines if the passed in object is equal to this object.
equals(Object) - Method in class com.ibm.crypto.fips.provider.**DHPrivateKey**
Determines if the passed in object is equal to this object.
equals(Object) - Method in class com.ibm.crypto.fips.provider.**DHPublicKey**
Determines if the passed in object is equal to this object.

F

FeedbackCipher - interface com.ibm.crypto.fips.provider.**FeedbackCipher**.
This interface represents the type of cipher that has a feedback mechanism built into it, such as CBC or CFB.
FIPSRuntimeException - exception com.ibm.crypto.fips.provider.**FIPSRuntimeException**.
Run time exception class.
FIPSRuntimeException() - Constructor for class com.ibm.crypto.fips.provider.**FIPSRuntimeException**
Constructs a FIPSRuntimeException with no detail message.
FIPSRuntimeException(String) - Constructor for class com.ibm.crypto.fips.provider.**FIPSRuntimeException**
Constructs a **FIPSRuntimeException** with the specified detail message.
fromFile(File) - Static method in class com.ibm.crypto.fips.provider.**IdentityDatabase**
Initialize an IdentityDatabase from file.
fromStream(InputStream) - Static method in class com.ibm.crypto.fips.provider.**IdentityDatabase**
Initialize an identity database from a stream.

G

getIdentity(PublicKey) - Method in class com.ibm.crypto.fips.provider.**IdentityDatabase**
Get an identity by key.
getIdentity(String) - Method in class com.ibm.crypto.fips.provider.**IdentityDatabase**
Get an identity named by the passed in string.
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.
getModulus() - Method in class com.ibm.crypto.fips.provider.**RSAPrivateKey**
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.**RSAPublicKey**
Return the modulus.
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.**DSAPublicKey**
Return the DSA parameters for the receiver.
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.
getPrimeExponentP() - Method in class com.ibm.crypto.fips.provider.**RSAPrivateCrtKey**
Returns the primeExponentP.
getPrimeExponentQ() - Method in class com.ibm.crypto.fips.provider.**RSAPrivateCrtKey**
Returns the primeExponentQ.
getPrimeP() - Method in class com.ibm.crypto.fips.provider.**RSAPrivateCrtKey**
Returns the primeP.
getPrimeQ() - Method in class com.ibm.crypto.fips.provider.**RSAPrivateCrtKey**
Returns the primeQ.
getPrivateExponent() - Method in class com.ibm.crypto.fips.provider.**RSAPrivateKey**
Return the private exponent.
getPrivateExponent() - Method in class com.ibm.crypto.fips.provider.**RSAPrivateCrtKey**
Return the private exponent.
getPublicExponent() - Method in class com.ibm.crypto.fips.provider.**RSAPrivateCrtKey**
Returns the public exponent.
getPublicExponent() - Method in class com.ibm.crypto.fips.provider.**RSAPublicKey**

H

I

IdentityDatabase - class com.ibm.crypto.fips.provider.**IdentityDatabase**.
An implementation of IdentityScope as a persistent identity database.
IdentityDatabase(File) - Constructor for class com.ibm.crypto.fips.provider.**IdentityDatabase**
Construct a new, empty database with a specified source file.
IdentityDatabase(String) - Constructor for class com.ibm.crypto.fips.provider.**IdentityDatabase**
Construct a new, empty database.
init() - Method in class com.ibm.crypto.fips.provider.**SHA**
Initialize the SHA information
init() - Method in class com.ibm.crypto.fips.provider.**MD5**
Initialize the MD5 information
init() - Method in class com.ibm.crypto.fips.provider.**SHA2**
Initialize the SHA5 information
init() - Method in class com.ibm.crypto.fips.provider.**SHA3**
Initialize the SHA2 information
init() - Method in class com.ibm.crypto.fips.provider.**SHA5**
Initialize the SHA3 information
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.**RSAKeyPairGenerator**
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) - Method in class com.ibm.crypto.fips.provider.**RSAKeyPairGenerator**
Initializes this key pair generator for a certain keysize.
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(int, SecureRandom) - Method in class com.ibm.crypto.fips.provider.**RSAKeyPairGenerator**
Initializes this KeyPairGenerator for given modulus and random source
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.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**X509Factory**
This function allows an application to verify the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DHParameterGenerator**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**SHA**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**SHA2**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**SHA3**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**SHA5**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**MD5**
This function allows an application to verify the algorithm is FIPS approved.
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.**DHParameters**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**AESParameters**
This function allows an application to verify 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.**AESKeyFactory**
This function allows an application to verify 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.**HmacSHA1KeyGenerator**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**SHA1withDSA**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**AESKeySpec**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DSAParameters**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**RSAKeyPairGenerator**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**CipherWithWrappingSpi**
This function allows an application to verify the algorithm is FIPS approved.

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DSAKeyFactory**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**RSAKeyFactory**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DSAKeyPairGenerator**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DESedeKeyGenerator**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**RSA**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**RSASSL**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DatawithRSA**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DatawithDSA**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**IdentityDatabase**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**SystemIdentity**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**AESCipher**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DESedeKeyFactory**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**SecureRandom**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DESedeParameters**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**SHA1withRSA**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**SystemSigner**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DESedeCipher**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DHKeyAgreement**

This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**HmacSHA1**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**AESKeyGenerator**
This function allows an application to verify the algorithm is FIPS approved.
isFipsApproved() - Method in class com.ibm.crypto.fips.provider.**DSAParameterGenerator**
This function allows an application to verify the algorithm is FIPS approved.
isFipsCertified() - Method in class com.ibm.crypto.fips.provider.**IBMJCEFIPS**
Method identifies if the cryptographic module is FIPS 140-2 certified
isFipsCertified() - Method in interface com.ibm.crypto.fips.provider.**ModuleStatus**
Method identifies if the cryptographic module is FIPS 140-2 certified
isFipsRunnable() - Static method in class com.ibm.crypto.fips.provider.**SelfTest**
Method identifies if the cryptographic module is FIPS 140-2 runable, in that the self test has completed with no failures.
isSelfTestInProgress() - Method in class com.ibm.crypto.fips.provider.**SelfTest**
Method identifies if a self test is currently in progress
isTrusted() - Method in class com.ibm.crypto.fips.provider.**SystemIdentity**
Is this identity trusted by sun.* facilities?
isTrusted() - Method in class com.ibm.crypto.fips.provider.**SystemSigner**
Returns true if this signer is trusted.

M

MD5 - class com.ibm.crypto.fips.provider.**MD5.**
The MD5 class is used to compute an MD5 message digest over a given buffer of bytes.
MD5() - Constructor for class com.ibm.crypto.fips.provider.**MD5**
Standard constructor, creates a new MD5 instance, allocates its buffers from the heap.
ModuleStatus - interface com.ibm.crypto.fips.provider.**ModuleStatus**.
This class is for determining the FIPS certification of the cryptographic module.

P

pad(byte[], int, int) - Method in interface com.ibm.crypto.fips.provider.**Padding**
Performs padding for the given data input.
Padding - interface com.ibm.crypto.fips.provider.**Padding**.
Padding interface.
padLength(int) - Method in interface com.ibm.crypto.fips.provider.**Padding**
Determines how long the padding will be for a given input length.

R

S

save() - Method in class com.ibm.crypto.fips.provider.**IdentityDatabase**
Saves the database to the default source file.
save(OutputStream) - Method in class com.ibm.crypto.fips.provider.**IdentityDatabase**
Save the database in its current state to an output stream.
SecureRandom - class com.ibm.crypto.fips.provider.**SecureRandom**.
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**
Constructs a new instance of this class.
SecureRandom(byte[]) - Constructor for class com.ibm.crypto.fips.provider.SecureRandom
Constructs a new instance of this class with a seed
SelfTest - class com.ibm.crypto.fips.provider.**SelfTest**.
This class tests the function of this cryptographic module.
SelfTest() - Constructor for class com.ibm.crypto.fips.provider.**SelfTest**
Constructs a new instance of this class.
setTrusted(boolean) - Method in class com.ibm.crypto.fips.provider.**SystemIdentity**
Set the trust status of this identity
SHA - class com.ibm.crypto.fips.provider.**SHA**.
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.
SHA1withDSA - class com.ibm.crypto.fips.provider.SHA1withDSA.
This class implements signature using SHA1 with DSA.
SHA1withDSA() - Constructor for class com.ibm.crypto.fips.provider.**SHA1withDSA**
Constructs a new instance of this class.
SHA1withRSA - class com.ibm.crypto.fips.provider.**SHA1withRSA**.
This class implements signature using SHA1 with RSA
SHA1withRSA() - Constructor for class com.ibm.crypto.fips.provider.**SHA1withRSA**
Construct a blank RSA object.
SHA2 - class com.ibm.crypto.fips.provider.**SHA2**.
This class implements the Secure Hash Algorithm 2 (SHA-256) 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.
SHA3 - class com.ibm.crypto.fips.provider.**SHA3**.

This class implements the Secure Hash Algorithm 3 (SHA-384) 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.
SHA5 - class com.ibm.crypto.fips.provider.**SHA5**.
This class implements the Secure Hash Algorithm 5 (SHA-512) developed by the National Institute of Standards and Technology along with the National Security Agency.
SHA5() - Constructor for class com.ibm.crypto.fips.provider.**SHA5**
Standard constructor, creates a new SHA5 instance, allocates its buffers from the heap.
size() - Method in class com.ibm.crypto.fips.provider.**IdentityDatabase**
Returns the number of identities in the database
SystemIdentity - class com.ibm.crypto.fips.provider.**SystemIdentity**.
An identity with a very simple trust mechanism.
SystemIdentity(String, IdentityScope) - Constructor for class com.ibm.crypto.fips.provider.**SystemIdentity**
Constructor for this class.
SystemSigner - class com.ibm.crypto.fips.provider.**SystemSigner**.
SunSecurity signer.
SystemSigner(String) - Constructor for class com.ibm.crypto.fips.provider.**SystemSigner**
Construct a signer with a given name.
SystemSigner(String, IdentityScope) - Constructor for class com.ibm.crypto.fips.provider.**SystemSigner**
Construct a signer with a name and a scope.

T

TDCNP - class com.ibm.crypto.fips.provider.**TDCNP.**
This class creates a DESede cipher with default mode CBC with no Padding.
TDCNP() - Constructor for class com.ibm.crypto.fips.provider.**TDCNP**
Creates an instance of DESede cipher with CBC mode and no Padding.
toString() - Method in class com.ibm.crypto.fips.provider.**RSAPrivateKey**
Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**SHA1withDSA**
Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**DatawithDSA**
Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**RSAPrivateCrtKey**

Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**IdentityDatabase**
Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**SystemIdentity**
Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**DHPrivateKey**
Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**SystemSigner**
Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**RSAPublicKey**
Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**DHPublicKey**
Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**DSAPublicKey**
Answers a string containing a concise, human-readable description of the receiver.
toString() - Method in class com.ibm.crypto.fips.provider.**DSAPrivateKey**
Returns a string containing a concise, human-readable description of the receiver.

U

unpad(byte[], int, int) - Method in interface com.ibm.crypto.fips.provider.**Padding**
Returns the index where padding starts.

X

X509Factory - class com.ibm.crypto.fips.provider.**X509Factory**.
This class defines a certificate factory for X.509 v3 certificates and X.509 v2 certificate revocation lists (CRLs).
X509Factory() - Constructor for class com.ibm.crypto.fips.provider.**X509Factory**
The constructor for this class.

Z

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.**DESedeKey**
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.**AESSecretKey**
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.**DHPrivateKey**
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.**DHPublicKey**
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.**DSAPublicKey**
This function zeroizes the key so that it isn't in memory
zeroize() - Method in class com.ibm.crypto.fips.provider.**DSAPrivateKey**
This function zeroizes the key so that it isn't in memory

Notices

Other company, product, and service names may be trademarks or service marks of others.