IBM Java JCE FIPS 140-2 Cryptographic Module

Security Policy

IBM JAVA JCE FIPS 140-2 Cryptographic Module
Revision: 1.6.1

February 2009

Status: Final

1.6.1 Edition (February 2009)
This edition applies to the 1.6.1 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 1.3.1) 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 1.2 to 1.3.1

The following was added to the 1.3.1 version of IBMJCEFIPS:

- The non-approved CTS mode has been added for both AES and Triple-DES.
- HMAC SHA-256, HMAC SHA-384 and HMAC SHA-512 support has been added.
- RSA Support for key sizes up to 16384 has been added
- RSA with SHA-256, SHA-384, and SHA-512 signature support added
- DH support for AES derived key added
- DH support for 2048 bit prime modulus added
- DSA support for 2048 bit modulus added
- TLS implementation added
- Auth Hmac for SHA256 truncated to 128 and SHA512 truncated to 256 added

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	Security Level 1

Authentication	
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

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 SP2 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, as well as, JVMs at the 6.x level, 5.x level, and 1.4.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.

The module supports the following approved algorithms:

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

	HMAC–SHA-256 HMAC–SHA-384 HMAC–SHA-512	
Random Number Generation	FIPS 186-2 Appendix 3.1	FIPS 186-2
Digital Signature	DSA (1024[1])	FIPS 186-2
Digital Signature	RSA (1024[2] – 16384[3])	PKCS#1 v1.5

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 (RSASSL) (Allowed in the Approved mode for key transport[4])
Key Agreement Primitives	Diffie-Hellman Shared Secret (256[5] –2048)	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

[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 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 and above 15360 bit are non-compliant for key transport.
[5] Note: Diffie-Hellman shared secret primitives provide between 40 and 112 bits of encryption strength.

Symmetric Cipher	AES	PCBC and CTS mode
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

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 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 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	Updates the cryptographic object with data to be encrypted/decrypted and returns the data in encrypted or decrypted form (based on the

MessageDigest.doFinal	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.

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.

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.

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.

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, HMAC-SHA1, HMAC-SHA 256, HMAC-SHA 384, and HMAC-SHA 512.

CSP/Key Name	Key Type	Generation/Input	Output	Storage	Zeroization	Roles
AES	AES	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
DSA	DSA	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
HMAC-SHA1	HMAC-SHA1	Generated internally using approved	Plaintext	RAM	Zeroized when zeroize() method is	Crypto Officer, Crypto User

		RNG			called	User
HMAC-SHA256	HMAC-SHA256	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
HMAC-SHA384	HMAC-SHA384	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
HMAC-SHA512	HMAC-SHA512	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
RSA	RSA	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
Triple-DES	Triple-DES	Generated internally using approved RNG	Plaintext	RAM	Zeroized when zeroize() method is called	Crypto Officer, Crypto User
Integrity Key	HMAC-SHA1	Hardcoded	Never	Harddisk	When the module is deleted	Crypto Officer
PRNG Seed Key	FIPS 186-2 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 Seed	Generated internally using non-approved RNG	Never	RAM	When a new seed is generated	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.

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.

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, HMAC SHA256, HMAC SHA384, HMAC SHA512 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 tests for both the FIPS-approved PRNG and the non FIPS-approved TRNG used to seed it as well as 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 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.

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 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 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.

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

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 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.

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.

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.

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

---

C

D

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.
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.
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

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

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() - 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(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(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`.

E

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

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

engineDigest() - Method in class com.ibm.crypto.fips.provider.SHA2
Compute the final hash and reset the engine so that it is ready for re-use, as specifed by MessageDigest.
engineDigest(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.SHA2
Compute the final hash and reset the engine so that it is ready for re-use, as specifed by MessageDigest.
engineDigest() - Method in class com.ibm.crypto.fips.provider.SHA3
Compute the final hash and reset the engine so that it is ready for re-use, as specifed by MessageDigest.
engineDigest(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.SHA3
Compute the final hash and reset the engine so that it is ready for re-use, as specifed by MessageDigest.
engineDigest() - Method in class com.ibm.crypto.fips.provider.SHA5
Compute the final hash and reset the engine so that it is ready for re-use, as specifed by MessageDigest.
engineDigest(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.SHA5
Compute the final hash and reset the engine so that it is ready for re-use, as specifed by MessageDigest.
engineDoFinal(byte[], int, int) - Method in class
com.ibm.crypto.fips.provider.AESCipher
Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation.
engineDoFinal(byte[], int, int, byte[], int) - Method in class
com.ibm.crypto.fips.provider.AESCipher
Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation.
engineDoFinal(byte[], int, int) - Method in class
com.ibm.crypto.fips.provider.DESedeCipher
Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation.
engineDoFinal(byte[], int, int, byte[], int) - Method in class
com.ibm.crypto.fips.provider.DESedeCipher
Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation.
engineDoFinal() - Method in class com.ibm.crypto.fips.provider.HmacSHA1
Completes the HMAC computation and resets the HMAC for further use, maintaining the secret key that the HMAC was initialized with.
engineDoFinal() - Method in class com.ibm.crypto.fips.provider.HmacSHA256
Completes the HMAC computation and resets the HMAC for further use, maintaining the secret key that the HMAC was initialized with.
engineDoFinal() - Method in class com.ibm.crypto.fips.provider.HmacSHA384

Generates an HMAC-SHA512 key.

engineGenerateParameters() - Method in class com.ibm.crypto.fips.provider.DHParameterGenerator

Generates the parameters.

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

Answers the newly generated parameters.

engineGeneratePrivate(KeySpec) - Method in class com.ibm.crypto.fips.provider.DHKeyFactory

Generates a private key object from the provided key specification (key material).

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

Generates a private key for the given key specification.

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

Generates a private key object from the provided key specification (key material).

engineGeneratePublic(KeySpec) - Method in class com.ibm.crypto.fips.provider.DHKeyFactory

Generates a public key object from the provided key specification (key material).

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

Generates a public key for the given key specification.

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

Generates a public key object from the provided key specification (key material).

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

Generates a `SecretKey` object from the provided key specification (key material).

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

Generates a `SecretKey` object from the provided key specification (key material).

engineGenerateSecret() - Method in class com.ibm.crypto.fips.provider.DHKeyAgreement

Generates the shared secret and returns it in a new buffer.

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

Generates the shared secret, and places it into the buffer `sharedSecret`, beginning at `offset`.

engineGenerateSecret(String) - Method in class com.ibm.crypto.fips.provider.DHKeyAgreement

Creates the shared secret and returns it as a secret key object of the requested algorithm type.

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

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

Returns the block size (in bytes).

engineGetBlockSize() - Method in class com.ibm.crypto.fips.provider.DESedeCipher
    Returns the block size (in bytes).
engineGetBlockSize() - Method in class com.ibm.crypto.fips.provider.RSA
    Returns the block size (in bytes).
engineGetBlockSize() - Method in class com.ibm.crypto.fips.provider.RSASSL
    Returns the block size (in bytes).
engineGetBlockSize() - Method in class com.ibm.crypto.fips.provider.TDCNP
    Returns the block size (in bytes).
engineGetDigestLength() - Method in class com.ibm.crypto.fips.provider.SHA
    Return the digest length in bytes
engineGetDigestLength() - Method in class com.ibm.crypto.fips.provider.SHA2
    Return the digest length in bytes
engineGetDigestLength() - Method in class com.ibm.crypto.fips.provider.SHA3
    Return the digest length in bytes
engineGetDigestLength() - Method in class com.ibm.crypto.fips.provider.SHA5
    Return the digest length in bytes
engineGetEncoded() - Method in class com.ibm.crypto.fips.provider.AESParameters

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

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

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

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

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

engineGetEncoded() - Method in class com.ibm.crypto.fips.provider.DSAParameters
    Returns the parameters in encoded bytes.
engineGetEncoded(String) - Method in class
com.ibm.crypto.fips.provider.DSAParameters
    Returns the parameters in encoded bytes with encoding method specified.
engineGetIV() - Method in class com.ibm.crypto.fips.provider.AESCipher
    Returns the initialization vector (IV) in a new buffer.
engineGetIV() - Method in class com.ibm.crypto.fips.provider.DESedeCipher
    Returns the initialization vector (IV) in a new buffer.
engineGetIV() - Method in class com.ibm.crypto.fips.provider.RSA
    Returns the initialization vector (IV) in a new buffer.
engineGetIV() - Method in class com.ibm.crypto.fips.provider.RSASSL
    Returns the initialization vector (IV) in a new buffer.
engineGetIV() - Method in class com.ibm.crypto.fips.provider.TDCNP
    Returns the initialization vector (IV) in a new buffer.

engineGetKeySize(Key) - Method in class com.ibm.crypto.fips.provider.AESCipher
Returns the key size of the given key object.
engineGetKeySize(Key) - Method in class com.ibm.crypto.fips.provider.DESedeCipher
Returns the key size of the given key object.
engineGetKeySize(Key) - Method in class com.ibm.crypto.fips.provider.RSA
Returns the key size of the given key object.
engineGetKeySize(Key) - Method in class com.ibm.crypto.fips.provider.RSASSL
Returns the key size of the given key object.
engineGetKeySize(Key) - Method in class com.ibm.crypto.fips.provider.TDCNP
Returns the key size of the given key object.
engineGetKeySpec(SecretKey, Class) - Method in class
com.ibm.crypto.fips.provider.AESKeyFactory
Returns a specification (key material) of the given key in the requested format.
engineGetKeySpec(SecretKey, Class) - Method in class
com.ibm.crypto.fips.provider.DESedeKeyFactory
Returns a specification (key material) of the given key in the requested format.
engineGetKeySpec(Key, Class) - Method in class
com.ibm.crypto.fips.provider.DHKeyFactory
Returns a specification (key material) of the given key object in the requested format.
engineGetKeySpec(Key, Class) - Method in class
com.ibm.crypto.fips.provider.DSAKeyFactory
Answers a key specification for a given key.
engineGetKeySpec(Key, Class) - Method in class
com.ibm.crypto.fips.provider.RSAKeyFactory
Returns a specification (key material) of the given key object in the requested format.
engineGetMacLength() - Method in class com.ibm.crypto.fips.provider.HmacSHA1
Returns the length of the HMAC in bytes.
engineGetMacLength() - Method in class com.ibm.crypto.fips.provider.HmacSHA256
Returns the length of the HMAC in bytes.
engineGetMacLength() - Method in class com.ibm.crypto.fips.provider.HmacSHA384
Returns the length of the HMAC in bytes.
engineGetMacLength() - Method in class com.ibm.crypto.fips.provider.HmacSHA512
Returns the length of the HMAC in bytes.
engineGetOutputSize(int) - Method in class com.ibm.crypto.fips.provider.AESCipher
Returns the length in bytes that an output buffer would need to be in order to hold the result of the next `update` or `doFinal` operation, given the input length `inputLen` (in bytes).
engineGetOutputSize(int) - Method in class com.ibm.crypto.fips.provider.DESedeCipher
Returns the length in bytes that an output buffer would need to be in order to hold the result of the next `update` or `doFinal` operation, given the input length `inputLen` (in bytes).
engineGetOutputSize(int) - Method in class com.ibm.crypto.fips.provider.RSA

Returns the length in bytes that an output buffer would need to be in order to hold the result of the next `update` or `doFinal` operation, given the input length `inputLen` (in bytes).

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

Returns the length in bytes that an output buffer would need to be in order to hold the result of the next `update` or `doFinal` operation, given the input length `inputLen` (in bytes).

engineGetOutputSize(int) - Method in class com.ibm.crypto.fips.provider.TDCNP

Returns the length in bytes that an output buffer would need to be in order to hold the result of the next `update` or `doFinal` operation, given the input length `inputLen` (in bytes).

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

Deprecated.

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

Have to implement Signature's abstract method engineGetParameter to be a concrete class.

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

Deprecated.

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

Have to implement Signature's abstract method engineGetParameter to be a concrete class.

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

Have to implement Signature's abstract method engineGetParameter to be a concrete class.

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

Have to implement Signature's abstract method engineGetParameter to be a concrete class.

engineGetParameter(String) - Method in class com.ibm.crypto.fips.provider.SHA5withRSA

Have to implement Signature's abstract method engineGetParameter to be a concrete class.

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

Returns the parameters used with this cipher.

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

Returns the parameters used with this cipher.

engineGetParameters() - Method in class com.ibm.crypto.fips.provider.RSA

Returns the parameters used with this cipher.

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

Returns the parameters used with this cipher.

engineGetParameters() - Method in class com.ibm.crypto.fips.provider.TDCNP

Returns the parameters used with this cipher.
engineGetParameterSpec(Class) - Method in class
com.ibm.crypto.fips.provider.AESParameters

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

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

engineGetParameterSpec(Class) - Method in class
com.ibm.crypto.fips.provider.DSAParameters
Return the parameter spec used by this parameter instance.
engineInit(int, Key, SecureRandom) - Method in class
com.ibm.crypto.fips.provider.AESCipher
Initializes this cipher with a key and a source of randomness.
engineInit(int, Key, AlgorithmParameterSpec, SecureRandom) - Method in class
com.ibm.crypto.fips.provider.AESCipher
Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.
engineInit(int, Key, AlgorithmParameters, SecureRandom) - Method in class
com.ibm.crypto.fips.provider.AESCipher
Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.
engineInit(SecureRandom) - Method in class
com.ibm.crypto.fips.provider.AESKeyGenerator
Initializes this key generator.
engineInit(AlgorithmParameterSpec, SecureRandom) - Method in class
com.ibm.crypto.fips.provider.AESKeyGenerator
Initializes this key generator with the specified parameter set and a user-provided source of randomness.
engineInit(int, SecureRandom) - Method in class
com.ibm.crypto.fips.provider.AESKeyGenerator
Initializes this key generator for a certain keysize, using the given source of randomness.
engineInit(AlgorithmParameterSpec) - Method in class
com.ibm.crypto.fips.provider.AESParameters

engineInit(byte[]) - Method in class com.ibm.crypto.fips.provider.AESParameters

engineInit(byte[], String) - Method in class com.ibm.crypto.fips.provider.AESParameters

engineInit(int, Key, SecureRandom) - Method in class
com.ibm.crypto.fips.provider.DESedeCipher
Initializes this cipher with a key and a source of randomness.

engineInit(Key, AlgorithmParameterSpec) - Method in class com.ibm.crypto.fips.provider.HmacSHA384
    Initializes the HMAC with the given secret key and algorithm parameters.
engineInit(SecureRandom) - Method in class com.ibm.crypto.fips.provider.HmacSHA384KeyGenerator
    Initializes this key generator.
engineInit(AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.HmacSHA384KeyGenerator
    Initializes this key generator with the specified parameter set and a user-provided source of randomness.
engineInit(int, SecureRandom) - Method in class com.ibm.crypto.fips.provider.HmacSHA384KeyGenerator
    Initializes this key generator for a certain keysize, using the given source of randomness.
engineInit(Key, AlgorithmParameterSpec) - Method in class com.ibm.crypto.fips.provider.HmacSHA512
    Initializes the HMAC with the given secret key and algorithm parameters.
engineInit(SecureRandom) - Method in class com.ibm.crypto.fips.provider.HmacSHA512KeyGenerator
    Initializes this key generator.
engineInit(AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.HmacSHA512KeyGenerator
    Initializes this key generator with the specified parameter set and a user-provided source of randomness.
engineInit(int, SecureRandom) - Method in class com.ibm.crypto.fips.provider.HmacSHA512KeyGenerator
    Initializes this key generator for a certain keysize, using the given source of randomness.
engineInit(int, Key, SecureRandom) - Method in class com.ibm.crypto.fips.provider.RSA
    Initializes this cipher with a key and a source of randomness.
engineInit(int, Key, AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.RSA
    Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.
engineInit(int, Key, AlgorithmParameters, SecureRandom) - Method in class com.ibm.crypto.fips.provider.RSA
    Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.
engineInit(int, Key, SecureRandom) - Method in class com.ibm.crypto.fips.provider.RSASSL
    Initializes this cipher with a key and a source of randomness.
engineInit(int, Key, AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.RSASSL
    Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.

engineInit(int, Key, AlgorithmParameters, SecureRandom) - Method in class com.ibm.crypto.fips.provider.RSASSL
    Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.
engineInit(int, Key, SecureRandom) - Method in class com.ibm.crypto.fips.provider.TDCNP
    Initializes this cipher with a key and a source of randomness.
engineInit(int, Key, AlgorithmParameterSpec, SecureRandom) - Method in class com.ibm.crypto.fips.provider.TDCNP
    Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.
engineInit(int, Key, AlgorithmParameters, SecureRandom) - Method in class com.ibm.crypto.fips.provider.TDCNP

engineInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.DatawithDSA
    Initialize the receiver with the specified private key, to be used for signing purposes.
engineInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.DatawithRSA
    Initialize the RSA object with a RSA private key.
engineInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.SHA1withDSA
    Initialize the receiver with the specified private key, to be used for signing purposes.
engineInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.SHA1withRSA
    Initialize the RSA object with a RSA private key.
engineInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.SHA2withRSA
    Initialize the RSA object with a RSA private key.
engineInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.SHA3withRSA
    Initialize the RSA object with a RSA private key.
engineInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.SHA5withRSA
    Initialize the RSA object with a RSA private key.
engineInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.DatawithDSA
    Initialize the receiver with the specified public key, to be used for verification purposes.
engineInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.DatawithRSA
    Initialize the RSA object with a RSA public key.
engineInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.SHA1withDSA

Initialize the receiver with the specified public key, to be used for verification purposes.
engineInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.SHA1withRSA
Initialize the RSA object with a RSA public key.
engineInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.SHA2withRSA
Initialize the RSA object with a RSA public key.
engineInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.SHA3withRSA
Initialize the RSA object with a RSA public key.
engineInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.SHA5withRSA
Initialize the RSA object with a RSA public key.
engineNextBytes(byte[]) - Method in class com.ibm.crypto.fips.provider.SecureRandom

engineReset() - Method in class com.ibm.crypto.fips.provider.HmacSHA1
Resets the HMAC for further use, maintaining the secret key that the HMAC was initialized with.
engineReset() - Method in class com.ibm.crypto.fips.provider.HmacSHA256
Resets the HMAC for further use, maintaining the secret key that the HMAC was initialized with.
engineReset() - Method in class com.ibm.crypto.fips.provider.HmacSHA384
Resets the HMAC for further use, maintaining the secret key that the HMAC was initialized with.
engineReset() - Method in class com.ibm.crypto.fips.provider.HmacSHA512
Resets the HMAC for further use, maintaining the secret key that the HMAC was initialized with.
engineReset() - Method in class com.ibm.crypto.fips.provider.SHA

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

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

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

engineSetMode(String) - Method in class com.ibm.crypto.fips.provider.AESCipher
Sets the mode of this cipher.
engineSetMode(String) - Method in class com.ibm.crypto.fips.provider.DESedeCipher
Sets the mode of this cipher.
engineSetMode(String) - Method in class com.ibm.crypto.fips.provider.RSA
Sets the mode of this cipher.
engineSetMode(String) - Method in class com.ibm.crypto.fips.provider.RSASSL
Sets the mode of this cipher.
engineSetMode(String) - Method in class com.ibm.crypto.fips.provider.TDCNP
Sets the mode of this cipher.

engineSetPadding(String) - Method in class com.ibm.crypto.fips.provider.AESCipher
Sets the padding mechanism of this cipher.
engineSetPadding(String) - Method in class com.ibm.crypto.fips.provider.DESedeCipher
Sets the padding mechanism of this cipher.
engineSetPadding(String) - Method in class com.ibm.crypto.fips.provider.RSA
Sets the padding mechanism of this cipher.
engineSetPadding(String) - Method in class com.ibm.crypto.fips.provider.RSASSL
Sets the padding mechanism of this cipher.
engineSetPadding(String) - Method in class com.ibm.crypto.fips.provider.TDCNP
Sets the padding mechanism of this cipher.
engineSetParameter(String, Object) - Method in class com.ibm.crypto.fips.provider.DatawithDSA
Deprecated. *Replaced with engineSetParameter(AlgorithmParameterSpec)*
engineSetParameter(String, Object) - Method in class com.ibm.crypto.fips.provider.DatawithRSA
Have to implement Signature's abstract method engineSetParameter to be a concrete class.
engineSetParameter(String, Object) - Method in class com.ibm.crypto.fips.provider.SHA1withDSA
Deprecated. *Replaced with engineSetParameter(AlgorithmParameterSpec)*
engineSetParameter(String, Object) - Method in class com.ibm.crypto.fips.provider.SHA1withRSA
Have to implement Signature's abstract method engineSetParameter to be a concrete class.
engineSetParameter(String, Object) - Method in class com.ibm.crypto.fips.provider.SHA2withRSA
Have to implement Signature's abstract method engineSetParameter to be a concrete class.
engineSetParameter(String, Object) - Method in class com.ibm.crypto.fips.provider.SHA3withRSA
Have to implement Signature's abstract method engineSetParameter to be a concrete class.
engineSetParameter(String, Object) - Method in class com.ibm.crypto.fips.provider.SHA5withRSA
Have to implement Signature's abstract method engineSetParameter to be a concrete class.
engineSetSeed(byte[]) - Method in class com.ibm.crypto.fips.provider.SecureRandom

engineSign() - Method in class com.ibm.crypto.fips.provider.DatawithDSA
Answers the signature bytes of the data updated so far.
engineSign() - Method in class com.ibm.crypto.fips.provider.DatawithRSA
Get message digest for all the data thus far updated, then sign the message digest.
engineSign() - Method in class com.ibm.crypto.fips.provider.SHA1withDSA
Answers the signature bytes of the data updated so far.
engineSign() - Method in class com.ibm.crypto.fips.provider.SHA1withRSA
Get message digest for all the data thus far updated, then sign the message digest.

engineSign() - Method in class com.ibm.crypto.fips.provider.SHA2withRSA
Get message digest for all the data thus far updated, then sign the message digest.
engineSign() - Method in class com.ibm.crypto.fips.provider.SHA3withRSA
Get message digest for all the data thus far updated, then sign the message digest.
engineSign() - Method in class com.ibm.crypto.fips.provider.SHA5withRSA
Get message digest for all the data thus far updated, then sign the message digest.
engineToString() - Method in class com.ibm.crypto.fips.provider.AESParameters

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

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

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

engineTranslateKey(SecretKey) - Method in class
com.ibm.crypto.fips.provider.AESKeyFactory
This action is not allowed in this provider.
engineTranslateKey(SecretKey) - Method in class
com.ibm.crypto.fips.provider.DESedeKeyFactory
This action is not allowed in this provider.
engineTranslateKey(Key) - Method in class com.ibm.crypto.fips.provider.DHKeyFactory
This action is not allowed in this provider.
engineTranslateKey(Key) - Method in class
com.ibm.crypto.fips.provider.DSAKeyFactory
This action is not allowed in this provider.
engineTranslateKey(Key) - Method in class
com.ibm.crypto.fips.provider.RSAKeyFactory
This action is not allowed in this provider.
engineUnwrap(byte[], String, int) - Method in class
com.ibm.crypto.fips.provider.AESCipher
Unwrap a previously wrapped key.
engineUnwrap(byte[], String, int) - Method in class
com.ibm.crypto.fips.provider.CipherWithWrappingSpi
Unwrap a previously wrapped key.
engineUnwrap(byte[], String, int) - Method in class
com.ibm.crypto.fips.provider.DESedeCipher
Unwrap a previously wrapped key.
engineUnwrap(byte[], String, int) - Method in class com.ibm.crypto.fips.provider.RSA
Unwrap a previously wrapped key.
engineUnwrap(byte[], String, int) - Method in class
com.ibm.crypto.fips.provider.RSASSL
Unwrap a previously wrapped key.
engineUnwrap(byte[], String, int) - Method in class
com.ibm.crypto.fips.provider.TDCNP
Unwrap a previously wrapped key.
engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.AESCipher

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

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

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

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

Update the bytes signed so far with the extra byte provided.

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

Update the bytes signed so far with the extra bytes provided.

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

Update a byte to be signed or verified.

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

Update an array of bytes to be signed or verified.

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

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

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

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

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

Processes the given byte.

engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.HmacSHA1

Processes the first `len` bytes in `input`, starting at `offset`.

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

Processes the given byte.

engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.HmacSHA256

Processes the first `len` bytes in `input`, starting at `offset`.

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

Processes the given byte.

engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.HmacSHA384

Processes the first `len` bytes in `input`, starting at `offset`.

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

Processes the given byte.

engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.HmacSHA512

Processes the first `len` bytes in `input`, starting at `offset`.

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

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

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

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

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

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

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

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

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

Update adds the passed byte to the digested data.

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

Update adds the selected part of an array of bytes to the digest.

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

Update the bytes signed so far with the extra byte provided.

engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.SHA1withDSA

Update the bytes signed so far with the extra bytes provided.

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

Update a byte to be signed or verified.

engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.SHA1withRSA

Update an array of bytes to be signed or verified.

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

Update adds the passed byte to the digested data.

engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.SHA2

Update adds the selected part of an array of bytes to the digest.

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

Update a byte to be signed or verified.

engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.SHA2withRSA

Update an array of bytes to be signed or verified.

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

Update adds the passed byte to the digested data.

engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.SHA3

Update adds the selected part of an array of bytes to the digest.

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

Update a byte to be signed or verified.

engineUpdate(byte[], int, int) - Method in class com.ibm.crypto.fips.provider.SHA3withRSA

Update an array of bytes to be signed or verified.

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

F

G

H

I

internalGenerateKey() - Method in class
com.ibm.crypto.fips.provider.DESedeKeyGenerator
    Generates the Triple-DES key.
internalGenerateKey() - Method in class
com.ibm.crypto.fips.provider.HmacSHA1KeyGenerator
    Generates an HMAC-SHA1 key.
internalGenerateKey() - Method in class
com.ibm.crypto.fips.provider.HmacSHA256KeyGenerator
    Generates an HMAC-SHA256 key.
internalGenerateKey() - Method in class
com.ibm.crypto.fips.provider.HmacSHA384KeyGenerator
    Generates an HMAC-SHA384 key.
internalGenerateKey() - Method in class
com.ibm.crypto.fips.provider.HmacSHA512KeyGenerator
    Generates an HMAC-SHA512 key.
internalGenerateKeyPair() - Method in class
com.ibm.crypto.fips.provider.DHKeyPairGenerator
    Generates a key pair.
internalGenerateKeyPair() - Method in class
com.ibm.crypto.fips.provider.DSAKeyPairGenerator

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

internalGenerateParameters() - Method in class
com.ibm.crypto.fips.provider.DHParameterGenerator
    Generates the parameters.
internalGenerateParameters() - Method in class
com.ibm.crypto.fips.provider.DSAParameterGenerator
    Answers the newly generated parameters.
internalGeneratePrivate(KeySpec) - Method in class
com.ibm.crypto.fips.provider.DHKeyFactory
    Generates a private key object from the provided key specification (key material).
internalGeneratePrivate(KeySpec) - Method in class
com.ibm.crypto.fips.provider.DSAKeyFactory
    Generates a private key for the given key specification.
internalGeneratePrivate(KeySpec) - Method in class
com.ibm.crypto.fips.provider.RSAKeyFactory
    Generates a private key object from the provided key specification (key material).
internalGeneratePublic(KeySpec) - Method in class
com.ibm.crypto.fips.provider.DHKeyFactory
    Generates a public key object from the provided key specification (key material).
internalGeneratePublic(KeySpec) - Method in class
com.ibm.crypto.fips.provider.DSAKeyFactory
    Generates a public key for the given key specification.

Returns the length in bytes that an output buffer would need to be in order to hold the result of the next `update` or `doFinal` operation, given the input length `inputLen` (in bytes).

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

Returns the length in bytes that an output buffer would need to be in order to hold the result of the next `update` or `doFinal` operation, given the input length `inputLen` (in bytes).

internalGetOutputSize(int) - Method in class com.ibm.crypto.fips.provider.RSA

Returns the length in bytes that an output buffer would need to be in order to hold the result of the next `update` or `doFinal` operation, given the input length `inputLen` (in bytes).

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

Deprecated.

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

Have to implement Signature's abstract method engineGetParameter to be a concrete class.

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

Deprecated.

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

Have to implement Signature's abstract method engineGetParameter to be a concrete class.

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

Have to implement Signature's abstract method engineGetParameter to be a concrete class.

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

Have to implement Signature's abstract method engineGetParameter to be a concrete class.

internalGetParameter(String) - Method in class com.ibm.crypto.fips.provider.SHA5withRSA

Have to implement Signature's abstract method engineGetParameter to be a concrete class.

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

Returns the parameters used with this cipher.

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

Returns the parameters used with this cipher.

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

Initialize the RSA object with a RSA private key.
internalInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.SHA1withDSA
Initialize the receiver with the specified private key, to be used for signing purposes.
internalInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.SHA1withRSA
Initialize the RSA object with a RSA private key.
internalInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.SHA2withRSA
Initialize the RSA object with a RSA private key.
internalInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.SHA3withRSA
Initialize the RSA object with a RSA private key.
internalInitSign(PrivateKey) - Method in class com.ibm.crypto.fips.provider.SHA5withRSA
Initialize the RSA object with a RSA private key.
internalInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.DatawithDSA
Initialize the receiver with the specified public key, to be used for verification purposes.
internalInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.DatawithRSA
Initialize the RSA object with a RSA public key.
internalInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.SHA1withDSA
Initialize the receiver with the specified public key, to be used for verification purposes.
internalInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.SHA1withRSA
Initialize the RSA object with a RSA public key.
internalInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.SHA2withRSA
Initialize the RSA object with a RSA public key.
internalInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.SHA3withRSA
Initialize the RSA object with a RSA public key.
internalInitVerify(PublicKey) - Method in class com.ibm.crypto.fips.provider.SHA5withRSA
Initialize the RSA object with a RSA public key.
internalNextBytes(byte[]) - Method in class com.ibm.crypto.fips.provider.SecureRandom

internalReset() - Method in class com.ibm.crypto.fips.provider.HmacSHA1
Resets the HMAC for further use, maintaining the secret key that the HMAC was initialized with.
internalReset() - Method in class com.ibm.crypto.fips.provider.HmacSHA256

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.DESedeKeyGenerator

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.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.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.DSAKeyFactory

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.DSAParameters

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.HmacSHA256

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.HmacSHA384KeyGenerator

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.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.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.SecureRandom

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.SHA1withRSA

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

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.SHA3withRSA

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

isFipsApproved() - Method in class com.ibm.crypto.fips.provider.SHA5withRSA

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

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

O

P

Q

R

S

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

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

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.

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.

SHA3withRSA - Class in com.ibm.crypto.fips.provider

This class implements the SHA1withRSA

SHA3withRSA() - Constructor for class com.ibm.crypto.fips.provider.SHA3withRSA

Construct a blank RSA object.

SHA5 - Class in com.ibm.crypto.fips.provider

This class implements the Secure Hash Algorithm 5 (SHA-5) 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.

SHA5withRSA - Class in com.ibm.crypto.fips.provider

This class implements the SHA1withRSA

SHA5withRSA() - Constructor for class com.ibm.crypto.fips.provider.SHA5withRSA

Construct a blank RSA object.

size() - Method in class com.ibm.crypto.fips.provider.IdentityDatabase

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

An identity with a very simple trust mechanism.

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

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

SunSecurity signer.

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

Construct a signer with a given name.

T

U

Z

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.DHPrivateKey
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.DSAPrivateKey
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.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.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.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.RSAPublicKey
This function zeroizes the key so that it isn't in memory.

Notices