RSA® BSAFE® Crypto-J JSAFE and JCE Software Module 6.2 and 6.2.1.1 Security Policy Level 1

This document is a non-proprietary security policy for RSA BSAFE Crypto-J JSAFE and JCE Software Module 6.2 and 6.2.1.1 (Crypto-J JSAFE and JCE Software Module) security software.

This document may be freely reproduced and distributed whole and intact including the copyright notice.

Note: Refer to the Change Summary for the location of the latest change to this document.

Contents:

Preface

This document is a non-proprietary security policy for the Crypto-J JSAFE and JCE Software Module from Dell Technologies[1].

This security policy describes how the Crypto-J JSAFE and JCE Software Module meets the Level 1 security requirements of FIPS 140-2.

Federal Information Processing Standards Publication 140-2 - Security Requirements for Cryptographic Modules (FIPS 140-2) details the U.S. Government requirements for cryptographic modules. More information about the FIPS 140-2 standard and validation program is available on the NIST website.

Terminology

In this document, the term *Crypto-J JSAFE and JCE Software Module* denotes the Crypto-J JSAFE and JCE Software Module 140-2 Security Level 1 validated Cryptographic Module.

The *Crypto-J JSAFE and JCE Software Module* is also referred to as:

- The Cryptographic Module
- The Java Crypto Module (JCM)
- The module.

Document Organization

This document explains the Crypto-J JSAFE and JCE Software Module features and functionality relevant to FIPS 140-2, and contains the following sections:

- This section, Preface provides an overview and introduction to the Security Policy.
- The Cryptographic Module, describes the module and how it meets the FIPS 140-2 Security Level 1 requirements.
- Secure Operation of the Module, addresses the required configuration for the FIPS 140-2 mode of operation.
- Acronyms, lists the definitions for the acronyms used in this document.

With the exception of the non-proprietary *RSA BSAFE Crypto-J JSAFE and JCE Software Module Security Policy* documents, the FIPS 140-2 Security Level 1 validation submission documentation is proprietary to Dell Technologies and is releasable only under appropriate non-disclosure agreements. For access to the documentation, please contact Dell Technologies

[1] Dell has acquired the BSAFE product line, which is now referred to as Dell BSAFE. Future module versions will be renamed to reflect this change.

1 The Cryptographic Module

This section provides an overview of the module, and contains the following topics:

- Introduction
- Module Characteristics
- Module Interfaces
- Roles and Services
- Cryptographic Key Management
- Cryptographic Algorithms
- Self-tests.

1.1 Introduction

More than a billion copies of the Dell BSAFE technology are embedded in today's most popular software applications and hardware devices. Encompassing one of the most widely-used and rich set of cryptographic algorithms as well as secure communications protocols, Dell BSAFE software is a set of complementary security products relied on by developers and manufacturers worldwide.

The Dell BSAFE™ Crypto-J (Crypto-J) software library relies on the JCM library. It includes a wide range of data encryption and signing algorithms, including AES, Triple-DES, the RSA Public Key Cryptosystem, the Elliptic Curve Cryptosystem, DSA, and the SHA-1 and SHA-2 message digest routines. Its software libraries, sample code and complete standards-based implementation enable near-universal interoperability for your networked and e-business applications.

1.2 Module Characteristics

The JCM is classified as a FIPS 140-2 multi-chip standalone module. As such, the JCM is tested on particular operating systems and computer platforms. The cryptographic boundary includes the JCM running on selected platforms that are running selected operating systems.

The JCM is validated for FIPS 140-2 Security Level 1 requirements.

The following table lists the certification levels sought for the JCM for each section of the FIPS 140-2 specification.

Table 1 Certification Levels

Section of the FIPS 140-2 Specification	Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	1
Overall	1

The JCM is packaged in a Java Archive (JAR) file containing all the code for the module.

The JCM API of the module is provided in the `jcmFIPS.jar` and `jcmandroidfips.jar` files.

The JCM is tested on the following platforms:

- Google™ Dalvik™ JRE 6.0 on Google Android™ 4.1.2 ARMv7 (32-bit) running on Google Nexus 7™ (Wi-Fi, 2012)
- Oracle® JRE 8.0 on Microsoft® Windows® 8.1 (64-bit) running on HP Envy 15
- OpenJDK 8.0 on CentOS 6.7 (64-bit) running on Dell™ PowerEdge™.

Compliance is maintained on platforms for which the binary executable remains unchanged. This includes, but is not limited to:

- Apple®
  - Mac OS® X 10.6 Snow Leopard®
    - x86 (32-bit), Apple® JDK 6.0/7.0
    - x86_64 (64-bit), Apple JDK 6.0/7.0.

- Canonical™
  - Ubuntu™ 12.04
    - x86 (32-bit), IBM® JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit® 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, OpenJDK 7u/8.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
  - Ubuntu 10.04
    - x86 (32-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
- Google
  - Android 2.1
    - ARM (32-bit) JDK 6.0
    - x86 (32-bit) JDK 6.0
  - Android 2.2
    - ARM (32-bit) JDK 6.0
    - x86 (32-bit) JDK 6.0
  - Android 2.3
    - ARM (32-bit) JDK 6.0
    - x86 (32-bit) JDK 6.0
  - Android 4.0
    - ARM (32-bit) JDK 6.0
    - x86 (32-bit) JDK 6.0
  - Android 4.1.2
    - ARM (32-bit) JDK 6.0
    - x86 (32-bit) JDK 6.0.
- HP
  - HP-UX 11.31
    - Itanium2 (32-bit), HP JRE 6.0/7.0
    - Itanium2 (64-bit), HP JRE 6.0/7.0.

- IBM
  - AIX® 6.1
    - PowerPC® (32-bit), IBM JRE 6.0/7.0
    - PowerPC (64-bit), IBM JRE 6.0/7.0.
  - AIX 7.1
    - PowerPC (32-bit), IBM JRE 6.0/7.0
    - PowerPC (64-bit), IBM JRE 6.0/7.0.
- Linux®
  - CentOS 6.7
    - x86_64 (64-bit), OpenJDK 8.0.
  - Novell® SUSE® Linux Enterprise Server 10
    - x86 (32-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - PowerPC (32-bit), IBM JRE 6.0/7.0
    - PowerPC (64-bit), IBM JRE 6.0/7.0.
  - Novell SUSE Linux Enterprise Server 11
    - x86 (32-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, OpenJDK 7u/8.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
    - PowerPC (32-bit), IBM JRE 6.0/7.0
    - PowerPC (64-bit), IBM JRE 6.0/7.0
    - Itanium 64-bit, Oracle JRE 6.0.
  - Novell SUSE Linux Enterprise Server 12
    - x86 (32-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
  - Red Hat® Enterprise Linux AS 5.0
    - PowerPC (32-bit), IBM JRE 6.0/7.0
    - PowerPC (64-bit), IBM JRE 6.0/7.0.

  - Red Hat Enterprise Linux 5.5, Security Enhanced Linux Configuration
    - x86 (32-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
  - Red Hat Enterprise Linux 6.0
    - x86 (32-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, OpenJDK 7u/8.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
  - Red Hat Enterprise Server 5.5
    - x86 (32-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, OpenJDK 7u, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
    - Itanium2 (64-bit), Oracle JRE 6.0/7.0/8.0.
- Microsoft
  - Windows Server 2003
    - x86 (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - Itanium (64-bit), Oracle JRE 6.0/7.0/8.0.
  - Windows Server 2008
    - x86 (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - Itanium (64-bit), Oracle JRE 6.0/7.0/8.0
  - Windows Server 2008 (SSLF configuration)
    - x86 (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
  - Windows Server 2012
    - x86 (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86_64 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0

- Windows Server 2012 R2
  - x86 (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
  - x86_64 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
- Windows Vista® (SSLF configuration)
  - x86 (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
  - x86_64 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
- Windows Vista Ultimate
  - x86 (32-bit), Oracle JRE 6.0/7.0/8.0, IBM JRE 6.0/7.0, Oracle JRockit 6.0
  - x86_64 (64-bit), Oracle JRE 6.0/7.0/8.0, IBM JRE 6.0/7.0, Oracle JRockit 6.0.
- Windows 8.1 Enterprise
  - x86 (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
  - x86_64 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
- Windows 8 Enterprise
  - x86 (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
  - x86_64 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.
- Windows 7 Enterprise SP1
  - x86 (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
  - x86_64 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0.

- Oracle
  - Solaris® 10
    - SPARC® v8+ (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0, Oracle JRockit 6.0
    - SPARC v9 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86 (32-bit), Oracle JRE 6.0/7.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0.

  - Solaris 11
    - SPARC v8+ (32-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0, Oracle JRockit 6.0
    - SPARC v9 (64-bit), IBM JRE 6.0/7.0, Oracle JRE 6.0/7.0/8.0, Oracle JRockit 6.0
    - x86 (32-bit), Oracle JRE 6.0/7.0.
- The FreeBSD Foundation
  - FreeBSD® 8.3 (64-bit) OpenJDK 7u.

Although porting is allowed on the listed platforms, the CMVP makes no claim as to the correct operation of the ported module. For a resolution on the issue of multi-user modes, see the NIST document Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program.

1.3 Module Interfaces

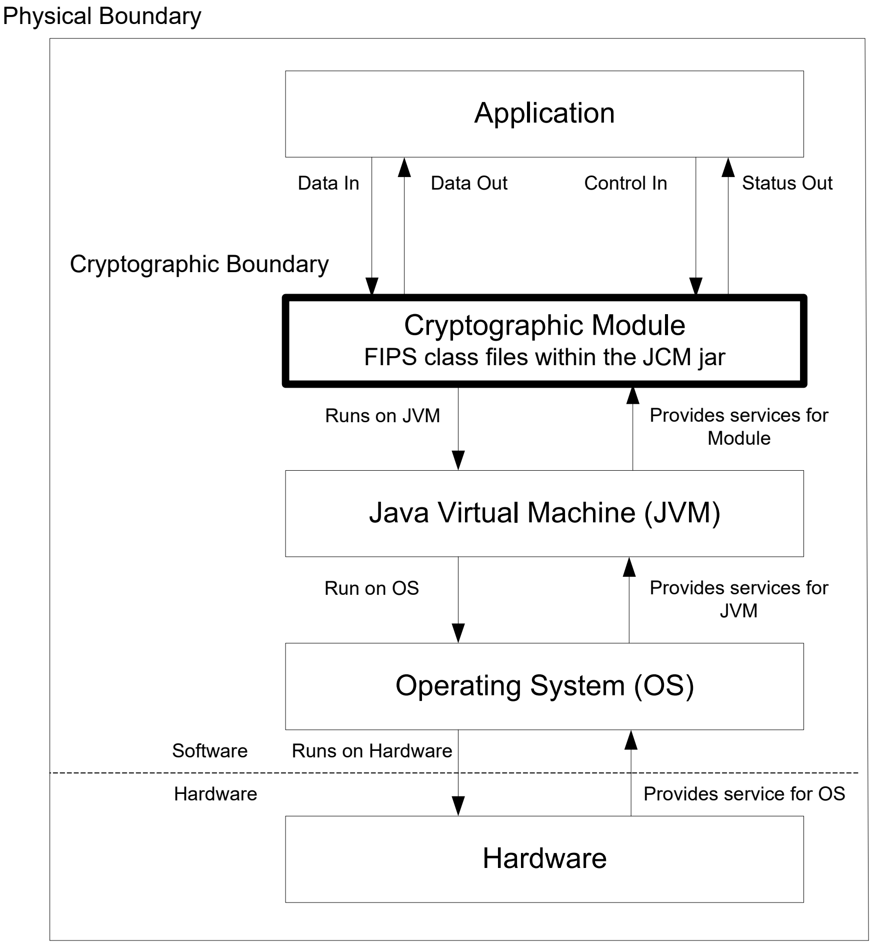
As a multi-chip standalone module, the physical interface to the JCM consists of a keyboard, mouse, monitor, serial ports and network adapters.

The underlying logical interface to the module is the API, documented in the relevant API *Javadoc*. The module provides the following four logical interfaces that have been designed within the module where “`input`” and “`output`” are indicated from the perspective of the module:

- Control Input - the invocation of all methods, the function and API names
- Data Input - input arguments to all constructors and methods specifying input parameters
- Data Output - modified input arguments, those passed by reference, and return values for all constructors and methods modifying input arguments and returning values
- Status Output - information returned by the methods and any exceptions thrown by constructors and methods.

This is shown in the following diagram.

Figure 1 JCM Logical Diagram



1.4 Roles and Services

The JCM is designed to meet all FIPS 140-2 Level 1 requirements, implementing both a Crypto Officer role and a Crypto User role. As allowed by FIPS 140-2, the JCM does not require user identification or authentication for these roles.

1.4.1 Crypto Officer Role

The Crypto Officer is responsible for installing and loading the module. Once the module has been installed and is operational, an operator can assume the Crypto Officer Role by constructing a `com.rsa.crypto.FIPS140Context` object by invoking the `ModuleConfig.getFIPS140Context(int mode, int role)` method with a role argument of `com.rsa.crypto.FIPS140Context.ROLE_CRYPTO_OFFICER`.

The Services section provides a list of services available to the Crypto Officer Role.

1.4.2 Crypto User Role

An operator can assume the Crypto User Role by constructing a `com.rsa.crypto.FIPS140Context` object by invoking the `ModuleConfig.getFIPS140Context(int mode, int role)` method with a role argument of `com.rsa.crypto.FIPS140Context.ROLE_USER`.

The Services section provides a list of services available to the Crypto User Role.

1.4.3 Services

The JCM provides services which are available for **both** FIPS 140-2 and non-FIPS 140-2 usage. For a list of FIPS 140-2 approved and FIPS 140-2 allowed algorithms, see Table 5.

The following table lists the un-authenticated services provided by the JCM which may be used by either Role, in either the FIPS or non-FIPS mode, in terms of the module interface. For each interface, lists of algorithms that are allowed and not allowed when operating the module in a FIPS 140-2 compliant way are specified.

Table 2 Services Available to the Crypto User and Crypto Officer Roles

Services Available to the Crypto User and Crypto Officer Roles	
Encryption/Decryption:	

`SymmCipher`	`clearSensitiveData`
	`clone`
	`doFinal`
	`getAlg`
	`getAlgorithmParams`
	`getBlockSize`
	`getCryptoModule`
	`getFeedbackSize`
	`getMaxInputLen`
	`getOutputSize`
	`init`
	`isIVRequired`
	`reInit`
	`update`

	Algorithms allowed for FIPS 140-2 usage
	AES (ECB, CBC, CFB, OFB, CTR, CCM, GCM, XTS) Triple-DES (ECB, CBC, CFB, OFB) PBE (PKCS #5 V2 - Approved for key storage)
	Algorithms not allowed for FIPS 140-2 usage
	AES (BPS, CBC_CS1, CBC_CS2, CBC_CS3) Triple-DES (CBC_CS1, CBC_CS2, CBC_CS3) DES DESX RC2® RC4® RC5® PBE (PKCS #12, PKCS #5, SSLCPBE)

Table 2 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Encryption/Decryption: (continued)	

`Cipher`	`clearSensitiveData`
	`clone`
	`doFinal`
	`getAlg`
	`getAlgorithmParams`
	`getBlockSize`
	`getCryptoModule`
	`getMaxInputLen`
	`getOutputSize`
	`init`
	`reInit`
	`update`

	Algorithms allowed for FIPS 140-2 usage
	RSA (Allowed for key transport) SP800-38F KW (AE, AD, AES-128, AES-192, AES-256) SP800-38F KWP (AE, AD, AES-128, AES-192, AES-256)
	Algorithms not allowed for FIPS 140-2 usage
	ECIES
Signature Generation/Verification:	

`Signature`	`clearSensitiveData`
	`clone`
	`getAlg`
	`getCryptoModule`
	`getSignatureSize`
	`initSign`
	`initVerify`
	`reInit`
	`sign`
	`update`
	`verify`

	Algorithms allowed for FIPS 140-2 usage
	RSA X9.31, PKCS #1 V.1.5, RSASSA-PSS DSA ECDSA
	Algorithms not allowed for FIPS 140-2 usage
	None

Table 2 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
MAC Generation/Verification:	

`MAC`	`clearSensitiveData`
	`clone`
	`getAlg`
	`getCryptoModule`
	`getMacLength`
	`init`
	`mac`
	`reset`
	`update`
	`verify`

	Algorithms allowed for FIPS 140-2 usage
	HMAC (When used with an approved Message Digest algorithm)
	Algorithms not allowed for FIPS 140-2 usage
	HMAC-MD5 PBHMAC (PKCS #12, PKIX)
Digest Generation:	

`MessageDigest`	`clearSensitiveData`
	`clone`
	`digest`
	`getAlg`
	`getCryptoModule`
	`getDigestSize`
	`reset`
	`update`

	Algorithms allowed for FIPS 140-2 usage
	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512 SHA-512/224 SHA-512/256
	Algorithms not allowed for FIPS 140-2 usage
	MD2 MD5 (Allowed in FIPS mode only for use in TLS) RIPEMD160

Table 2 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Parameters:	

`AlgInputParams`	`clone`
	`get`
	`getCryptoModule`
	`set`

`AlgorithmParams`	`getCryptoModule`

`DHParams`	`getCounter`
	`getCryptoModule`
	`getG`
	`getJ`
	`getMaxExponentLen`
	`getP`
	`getQ`
	`getSeed`

`DomainParams`	`getCryptoModule`

`DSAParams`	`getCounter`
	`getCryptoModule`
	`getDigestAlg`
	`getG`
	`getP`
	`getQ`
	`getSeed`

`ECParams`	`getA`
	`getB`
	`getBase`
	`getBaseDigest`
	`getBaseSeed`
	`getCofactor`
	`getCryptoModule`
	`getCurveName`
	`getDigest`
	`getFieldMidTerms`
	`getFieldPrime`
	`getFieldSize`
	`getFieldType`
	`getOrder`
	`getSeed`
	`getVersion`

`ECPoint`	`clearSensitiveData`
	`getEncoded`
	`getX`
	`getY`

Table 2 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Parameters: (continued)	

`PQGParams`	`getCryptoModule`
	`getG`
	`getP`
	`getQ`

Parameter Generation:	

`AlgParamGenerator`	`generate`
	`getCryptoModule`
	`initGen`
	`initVerify`
	`verify`

	Algorithms allowed for FIPS 140-2 usage
	EC DSA Diffie-Hellman
	Algorithms not allowed for FIPS 140-2 usage
	None
Key Establishment Primitives:	

`KeyAgreement`	`clearSensitiveData`
	`clone`
	`doPhase`
	`getAlg`
	`getCryptoModule`
	`getSecret`
	`init`

	Algorithms allowed for FIPS 140-2 usage
	Diffie-Hellman (primitives only) EC Diffie-Hellman (primitives only)
	Algorithms not allowed for FIPS 140-2 usage
	None

Table 2 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Key Generation:	

`KeyGenerator`	`clearSensitiveData`
	`generate`
	`getCryptoModule`
	`initialize`

	Algorithms allowed for FIPS 140-2 usage
	AES Triple-DES
	Algorithms not allowed for FIPS 140-2 usage
	DES DESX RC2 RC4 RC5 Shamir Secret Sharing

`KeyPairGenerator`	`clearSensitiveData`
	`clone`
	`generate`
	`getCryptoModule`
	`initialize`

	Algorithms allowed for FIPS 140-2 usage
	EC RSA DSA Diffie-Hellman
	Algorithms not allowed for FIPS 140-2 usage
	RSA Keypair Generation MultiPrime
Keys:	

`DHPrivateKey`	`clearSensitiveData`
	`clone`
	`getAlg`
	`getCryptoModule`
	`getParams`
	`getX`

Table 2 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Keys: (continued)	
DHPublicKey	clearSensitiveData clone getAlg getCryptoModule getParams getY isValid
DSAPrivateKey	clearSensitiveData clone getAlg getCryptoModule getParams getX
DSAPublicKey	clearSensitiveData clone getAlg getCryptoModule getParams getY isValid
ECPrivateKey	clearSensitiveData clone getAlg getCryptoModule getD getParams
ECPublicKey	clearSensitiveData clone getAlg getCryptoModule getParams getPublicPoint isValid
Key	clearSensitiveData clone getAlg getCryptoModule

Table 2 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Keys: (continued)	
KeyBuilder	newDHParams newDHPrivateKey newDHPublicKey newDSAParams newDSAPrivateKey newDSAPublicKey newECParams newECPrivateKey newECPublicKey newPasswordKey newPQGParams newRSAPrivateKey newRSAPublicKey newSecretKey recoverShamirSecretKey
KeyPair	clearSensitiveData clone getAlgorithm getPrivate getPublic
PasswordKey	clearSensitiveData clone getAlg getCryptoModule getKeyData getPassword
PrivateKey	clearSensitiveData clone getAlg getCryptoModule
PublicKey	clearSensitiveData clone getAlg getCryptoModule isValid

Table 2 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Keys: (continued)	
RSAPrivateKey	clearSensitiveData clone getAlg getCoeff getCryptoModule getD getE getExpP getExpQ getN getOtherMultiPrimeInfo getP getQ hasCRTInfo isMultiprime
RSAPublicKey	clearSensitiveData clone getAlg getCryptoModule getE getN isValid
SecretKey	clearSensitiveData getAlg getCryptoModule getKeyData
SharedSecretKey	clearSensitiveData clone getAlg getCryptoModule getKeyData getSharedParams

Table 2 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Key Derivation:	

`KDF`	`clearSensitiveData`
	`clone`
	`generate`
	`getCryptoModule`

	Algorithms allowed for FIPS 140-2 usage
	PBKDF2 (Approved for key storage) KDFTLS10 (For use with TLS versions 1.0 and 1.1) KDFTLS12 (For use with TLS version 1.2)
	Algorithms not allowed for FIPS 140-2 usage
	PKCS #5 KDF PKCS #12 KDF scrypt
Random Number Generation:	

`SecureRandom`	`autoseed`
	`clearSensitiveData`
	`getCryptoModule`
	`newInstance`
	`nextBytes`
	`selfTest`
	`setAlgorithmParams`
	`setOperationalParameters`
	`setSeed`

	Algorithms allowed for FIPS 140-2 usage
	CTR DRBG Hash DRBG HMAC DRBG
	Algorithms not allowed for FIPS 140-2 usage
	FIPS 186-2 PRNG (Change Notice General)
Other Services:	
`BigNum`	`getBitLength` `toOctetString`

Table 2 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Other Services: (continued)	
CryptoModule	getDeviceType getKeyBuilder getModuleOperations newAlgInputParams newAlgParamGenerator newAsymmetricCipher newKDF newKeyAgreement newKeyGenerator newKeyPairGenerator newKeyWrapCipher newMAC newMessageDigest newSecureRandom newSignature newSymmetricCipher
JCMCloneable	clone
ModuleConfig	getEntropySource getFIPS140Context getSecurityLevel getVersionDouble getVersionString initFIPS140RolePINs isFIPS140Compliant newCryptoModule setEntropySource
ModuleLoader	load
ModuleOperations	perform
PasswordKey	clearSensitiveData clone getAlg getCryptoModule getKeyData getPassword
SelfTestEvent	getTestId getTestName
SelfTestEventListener	failed finished passed started
SensitiveData	clearSensitiveData

For more information on each function, see the relevant API *Javadoc*.

1.5 Cryptographic Key Management

1.5.1 Key Generation

The module supports the generation of the DSA, RSA, and Diffie-Hellman (DH) and ECC public and private keys. In the FIPS-Approved mode, RSA keys can only be generated using the Approved 186-4 RSA key generation method.

The module also employs a FIPS-Approved AES Counter-mode DRBG (AES-128-CTR DRBG) for generating asymmetric and symmetric keys used in algorithms such as AES, Triple-DES, RSA, DSA, DH and ECC.

1.5.2 Key Protection

All key data resides in internally allocated data structures and can only be output using the JCM API. The operating system and the JRE safeguards memory and process space from unauthorized access.

1.5.3 Key Zeroization

The module stores all its keys and Critical Security Parameters (CSPs) in volatile memory. Users can ensure CSPs are properly zeroized by making use of the `<object>.clearSensitiveData()` method. All of the module's keys and CSPs are zeroizable. For more information about clearing CSPs, see the relevant API *Javadoc*.

1.5.4 Key Storage

The JCM does not provide long-term cryptographic key storage. Storage of keys is the responsibility of the user. The Crypto User and Crypto Officer roles have equal and complete access to all keys and CSPs.

The following table shows how the storage of keys and CSPs are handled:

Table 3 Key and CSP Storage

Item	Storage
AES keys	In volatile memory only (plaintext)
Triple-DES keys	In volatile memory only (plaintext)
HMAC with SHA-1 and SHA-2 keys (SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256)	In volatile memory only (plaintext)
ECC private keys/public key	In volatile memory only (plaintext)
Diffie-Hellman private key/public key	In volatile memory only (plaintext)
RSA private key/public key	In volatile memory only (plaintext)

Table 3 Key and CSP Storage (continued)

Item	Storage
DSA private key/public key	In volatile memory only (plaintext)
CTR DRBG Entropy	In volatile memory only (plaintext)
CTR DRBG V Value	In volatile memory only (plaintext)
CTR DRBG Key	In volatile memory only (plaintext)
CTR DRBG init_seed	In volatile memory only (plaintext)
Hash DRBG Entropy	In volatile memory only (plaintext)
Hash DRBG V Value	In volatile memory only (plaintext)
Hash DRBG C	In volatile memory only (plaintext)
Hash DRBG init_seed	In volatile memory only (plaintext)
HMAC DRBG Entropy	In volatile memory only (plaintext)
HMAC DRBG V Value	In volatile memory only (plaintext)
HMAC DRBG Key	In volatile memory only (plaintext)
HMAC DRBG init_seed	In volatile memory only (plaintext)
TLS Pre-Master Secret	In volatile memory only (plaintext)
TLS Master Secret	In volatile memory only (plaintext)
TLS Session Keys	In volatile memory only (plaintext)

1.5.5 Key Access

An authorized operator of the module has access to all key data created during JCM operation.

Note: The User and Officer roles have equal and complete access to all keys.

The following table lists the different services provided by the module with the type of access to keys or CSPs.

Table 4 Key and CSP Access

Service	Key or CSP	Type of Access
Asymmetric encryption and decryption	Asymmetric keys (RSA)	Read/Execute
Encryption and decryption	Symmetric keys (AES, Triple-DES)	Read/Execute
Digital signature and verification	Asymmetric keys (DSA, RSA, ECDSA)	Read/Execute
Hashing	None	N/A
MAC	HMAC keys	Read/Execute
Random number generation	CTR DRBG entropy, V, key, init_seed Hash DRBG entropy, V, C, init_seed HMAC DRBG entropy, V, key, init_seed	Read/Write/Execute
Key derivation	TLS Pre-Master Secret TLS Master Secret TLS Session Keys	Read/Execute
Key establishment primitives	Asymmetric keys (DH, ECDH)	Read/Execute
Key generation	Symmetric keys (AES, Triple-DES) Asymmetric keys (DSA, EC DSA, RSA, DH, ECDH) MAC keys (HMAC)	Write
Self-test	Hard-coded keys, (AES, Triple-DES, RSA, DSA, ECDSA, HMAC) Hard-coded entropy, strength, and seed (HMAC DRBG)	Read/Execute
Show status	None	N/A
Zeroization	All	Read/Write

1.6 Cryptographic Algorithms

The JCM offers a wide range of cryptographic algorithms. This section describes the algorithms that can be used when operating the module in a FIPS 140-2 compliant manner.

The following table lists the FIPS 140-2 approved and FIPS 140-2 allowed algorithms that can be used when operating the module in a FIPS 140-2 compliant way.

Table 5 JCM FIPS 140-2 Approved Algorithms

Algorithm Type	Algorithm	Validation Certificate
Asymmetric Cipher	RSA Key establishment methodology provides 112 or 128 bits of encryption strength.	Non-Approved (Allowed in FIPS mode for key transport*)
Key Agreement Primitives	Diffie-Hellman (primitives only, 2048 to 4096 bit public key sizes, 160 to 4096 bit private key sizes) EC Diffie-Hellman (primitives only, 224 to 571 bit key sizes)	Non-Approved (Allowed in FIPS mode) CVL (Cert. #1024)
Key Derivation	PBKDF2	Vendor Affirmed (Approved in FIPS mode for key storage**)
	KDFTLS10***	Certificate #471
	KDFTLS12**** with SHA-256, SHA-384, SHA-512	Certificate #471
Key Wrap SP800-38F	KW (AE, AD, AES-128, AES-192, AES-256) KWP (AE, AD, AES-128, AES-192, AES-256)	Certificate #3263
Message Authentication Code	HMAC*****	Certificate #2062
Message Digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	Certificate #2701
	SHA-512/224, SHA-512/256	Certificate #2701
Random Bit Generator	CTR DRBG	Certificate #722
	Hash DRBG	Certificate #722
	HMAC DRBG	Certificate #722
Signature*****	RSA X9.31, PKCS #1 V.1.5, RSASSA-PSS (2048 and 3072 bit key sizes)	Certificate #1663
	DSA (2048 and 3072 bit key sizes)	Certificate #932
	ECDSA (224 to 571 bit key sizes)	Certificate #619

Table 5 JCM FIPS 140-2 Approved Algorithms (continued)

Algorithm Type	Algorithm	Validation Certificate
Symmetric Cipher	AES (ECB, CBC, CFB, OFB, CTR, CCM, GCM, XTS) [128, 192, 256 bit key sizes]	Certificate #3263
	Triple-DES****** (ECB, CBC, CFB, OFB)	Certificate #1852

*The module implements RSA encrypt/decrypt, which is non-Approved. A calling application may use this to implement a key transport scheme, which is allowed for use in FIPS mode.

**The module implements PBKDF2 as the PBKDF algorithm as defined in SP800-132. This can be used in FIPS mode when used with a FIPS-approved Symmetric Cipher and Message Digest algorithm.
For information on how to use PBKDF, see The following restrictions apply to the use of PBKDF:.

***Key Derivation in TLS for versions 1.0 and 1.1. The TLS protocol has not been tested by the CAVP and CMVP.

****Key Derivation in TLS for versions 1.2. The TLS protocol has not been tested by the CAVP and CMVP.

*****When used with a FIPS-approved Message Digest algorithm.

******For information on the restrictions applicable to the use of two-key Triple-DES, see The following restrictions apply to the use of Triple-DES:.

The following lists all other available algorithms in the JCM that are **not allowable** for FIPS 140-2 usage. These algorithms must not be used when operating the module in a FIPS 140-2 compliant way.

- AES in BPS mode for FPE
- AES in CBC_CS1, CBC_CS2 or CBC_CS3 mode for CTS
- DES
- DESX
- ECIES
- FIPS 186-2 PRNG (Change Notice General)
- HMAC-MD5
- MD2
- MD5[2]
- PKCS #5 KDF
- PKCS #12 KDF
- RC2 block cipher
- RC4 stream cipher
- RC5 block cipher
- RSA Keypair Generation MultiPrime (2 or 3 primes)
- RIPEMD160
- scrypt
- Shamir Secret Sharing
- Triple-DES in CBC_CS1, CBC_CS2 or CBC_CS3 mode for CTS.

[2] MD5 is allowed in FIPS mode only for use in TLS.

1.7 Self-tests

The module performs power-up and conditional self-tests to ensure proper operation.

If the power-up self-test fails, the module is disabled and throws a `SecurityException`. The module cannot be used within the current JVM.

If the conditional self-test fails, the module throws a `SecurityException` and aborts the operation. A conditional self-test failure does NOT disable the module.

1.7.1 Power-up Self-tests

The following FIPS 140-2 required power-up self-tests are implemented in the module:

- AES Decrypt KAT
- AES Encrypt KAT
- Triple-DES Decrypt KAT
- Triple-DES Encrypt KAT
- SHA-512 KAT
- KDFTLS10 KAT
- KDFTLS12 SHA-256 KAT
- Hash DRBG Self-Test
- CTR DRBG Self-Test
- HMAC DRBG Self-Test
- HMAC-SHA1 software integrity check
- RSA Signature Generation KAT
- RSA Signature Verification KAT
- DSA Sign/Verify Test
- ECDSA Sign/Verify Test.

Power-up self-tests are executed automatically when the module is loaded into memory.

1.7.2 Conditional Self-tests

The module performs two conditional self-tests:

- Pair-wise Consistency Tests each time the module generates a DSA, RSA or EC public/private key pair.
- Continuous RNG (CRNG) Test each time the module produces random data, as per the FIPS 140-2 standard. The CRNG test is performed on all approved and non-approved PRNGs (HMAC DRBG, HASH DRBG, CTR DRBG, FIPS186 PRNG, non-approved entropy source).

1.7.3 Mitigation of Other Attacks

RSA key operations implement blinding by default. Blinding is a reversible way of modifying the input data, so as to make the RSA operation immune to timing attacks. Blinding has no effect on the algorithm other than to mitigate attacks on the algorithm.

RSA Blinding is implemented through blinding modes, for which the following options are available:

- Blinding mode off
- Blinding mode with no update, where the blinding value is squared for each operation
- Blinding mode with full update, where a new blinding value is used for each operation.

2 Secure Operation of the Module

The following guidance must be followed in order to operate the module in a FIPS 140-2 mode of operation, in conformance with FIPS 140-2 requirements.

Note: The module operates as a Validated Cryptographic Module only when the rules for secure operation are followed.

2.1 Module Configuration

To operate the module in compliance with FIPS 140-2 Level 1 requirements, the module must be loaded using the following method:

```
com.rsa.crypto.jcm.ModuleLoader.load()
```

The `ModuleLoader.load()` method extracts arguments from the `com.rsa.cryptoj.common.jcm.JavaModuleProperties` class, which is created using the `com.rsa.cryptoj.common.jcm.CryptoJModulePropertiesFactory` class.

The following arguments are extracted:

- For the Android platform:
  - The module jar file
- For all platforms:
  - The security level, specified as the constant `ModuleConfig.LEVEL_1`. This should have a value of 1.
  - The `SelfTestEventListener` to use.

Using the specified `securityLevel` ensures that the module is loaded for use in a FIPS 140-2 Level 1 compliant way.

Once the load method has been successfully called, the module is operational.

2.2 Security Roles, Services and Authentication Operation

To assume a role once the module is operational, construct a `FIPS140Context` object for the desired role using the `FIPS140Context.getFIPS140Context(int mode, int role)` method. This object can then be used to perform cryptographic operations using the module.

The mode argument must be the value `FIPS140Context.MODE_FIPS140`.

The available role values are the constants `FIPS140Context.ROLE_CRYPTO_OFFICER` and `FIPS140Context.ROLE_USER`.

No role authentication is required for operation of the module in Security Level 1 mode. When in Security Level 1 mode, invocation of methods which are particular to Security Level 2 Roles, Services and Authentication will result in an error.

2.3 Crypto User Guidance

This section provides guidance to the module user to ensure that the module is used in a FIPS 140-2 compliant way.

Section 2.3.1 provides algorithm-specific guidance. The requirements listed in this section are not enforced by the module and must be ensured by the module user.

Section 2.3.2 provides guidance on obtaining assurances for Digital Signature Applications.

Section 2.3.3 provides guidance on the entropy requirements for secure key generation.

Section 2.3.4 provides general crypto user guidance.

2.3.1 Crypto User Guidance on Algorithms

- The Crypto User must only use algorithms approved for use in a FIPS 140-2 mode of operation, as listed in Table 5, JCM FIPS 140-2 Approved Algorithms.
- Only FIPS 140-2 Approved RNGs may be used for generation of keys (asymmetric and symmetric).
- When using an approved RNG, the number of bytes of seed key input must be equivalent to or greater than the security strength of the keys the caller wishes to generate. For example, a 256-bit or higher seed key input when generating 256-bit AES keys.
- When using an Approved RNG to generate keys or DSA parameters, the RNG's requested security strength must be at least as great as the security strength of the key being generated. That means that an Approved RNG with an appropriate strength must be used. For more information on requesting the RNG security strength, see the relevant API *Javadoc*.
- FIPS 186-2 RNG is not to be used in an approved FIPS 140-2 mode of operation.
- In case the power to the module is lost and then restored, the key used for the AES GCM encryption/decryption shall be re-distributed.
- When generating key pairs using the `KeyPairGenerator` object, the `generate(boolean pairwiseConsistency)` method must not be invoked with an argument of `false`. Use of the no-argument `generate()` method is recommended.

- When using GCM mode for symmetric encryption, the authentication tag length and authenticated data length may be specified as input parameters, but the Initialization Vector (IV) must not be specified, except for use within the TLS protocol where the IV generation mechanism defined in RFC 5288 must be used.

  When generated internally, the GCM IV is constructed deterministically according to Section 8.2.1 of SP 800-38D such that:

  – the fixed field (module name) is a 32-bit value derived from the runtime memory address of a Java class within the module

  – the invocation field is a 64-bit integer counter that is initialized on startup to a value consisting of the current time (as milliseconds since the Epoch, 44 bits) concatenated with 22 bits of zeros - that is, (time in milliseconds) x 2^{22}.

- For RSASSA-PSS: If `nLen` is 1024 bits, and the output length of the **approved** hash function output block is 512 bits, then the length of the salt (`sLen`) **shall** be `0<=sLen<=hLen - 2`
  Otherwise, the length of the salt **shall** be `0 <=sLen<=hLen` where `hLen` is the length of the hash function output block (in bytes or octets).

- EC key pairs must have named curve domain parameters from the set of NIST-recommended named curves: P-224, P-256, P-384, P-521, B-233, B-283, B-409, B-571, K-233, K-283, K-409, and K-571. Named curves P192, B163, and K163 are allowed for legacy-use. The domain parameters can be specified by name or can be explicitly defined.

- EC Diffie-Hellman primitives must use curve domain parameters from the set of NIST recommended named curves listed above. The domain parameters can be specified by name, or can be explicitly defined. Using the NIST-recommended curves, the computed Diffie-Hellman shared secret provides between 112 bits and 256 bits of security.

- The bit length of the input parameter to the Diffie-Hellman primitives method must be at least 2048 bits. 2048 bit Diffie Hellman shared secret provides 112 bits of encryption strength.

- DSA parameters shall be generated according to FIPS 186-4 by specifying the algorithm string “DSA” when creating the `AlgParamGenerator` object. The non-Approved algorithm specified by the string “`PQG`” shall not be used.

- The length of a DSA modulus, P value, used to create a key pair for digital signature generation and verification must be either 2048 or 3072 bits. For digital signature verification, 1024 bits is allowed for legacy-use.

- For RSA digital signature generation, an Approved RNG must be used.

- RSA keys used for signing shall not be used for any other purpose other than digital signatures.

- RSA key pairs shall be generated according to FIPS 186-4 by specifying a `KEY_TYPE` parameter of 0. This is the default `KEY_TYPE` value, so may be omitted as an input parameter (to the `KeyPairGenerator.initialize` method).

- The length of an RSA key pair for digital signature generation and verification must be 2048 or 3072 bits. For digital signature verification, 1024 bits is allowed for legacy-use. RSA keys shall have a public exponent of at least 65537.

- SHA1 is disallowed for the generation of digital signatures.
- The key length for an HMAC generation or verification must be between 112 and 4096 bits, inclusive. For HMAC verification, a key length greater than or equal to 80 and less than 112 is allowed for legacy-use.
- The following restrictions apply to the use of PBKDF:
  - The minimum password length is 14 characters, which has a strength of approximately 112 bits, assuming a randomly selected password using the extended ASCII printable character set is used.
    For random passwords - a string of characters from a given set of characters in which each character is equally likely to be selected - the strength of the password is given by: `S=L*(log N/log 2)` where N is the number of possible characters (for example, ASCII printable characters `N = 95`, extended ASCII printable characters `N = 218`) and L is the number of characters. A password of the strength S can be guessed at random with the probability of $1/2^S$.
  - Keys generated using PBKDF shall only be used in data storage applications.
  - The length of the randomly-generated portion of the salt shall be at least 16 bytes. For more information see `nist-sp800-132.pdf`.
  - The iteration count shall be selected as large as possible, a minimum of 1000 iterations is recommended.
  - The maximum key length is $(2^{32} -1)*b$, where b is the digest size of the hash function.
  - The key derived using PBKDF can be used as referred to in SP800-132, Section 5.4, option 1 and 2.
- The following restrictions apply to the use of Triple-DES:
  - The use of three-key Triple-DES is approved beyond 2013 without restriction.
  - The use of two-key Triple-DES is approved beyond 2013. Until 31 December 2015, two-key Triple-DES is allowed with the restriction that at most 2^{20} blocks of data can be encrypted with the same key.
  - The use of two-key Triple-DES is disallowed beyond 2015. Two-key Triple-DES can be used to decrypt ciphertext for legacy use after 2015.

    For more information about the use of two-key Triple-DES, see NIST Special Publication 800-131A “Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths”.

- The JCM is affected by CVE-2019-3738 (Missing Required Cryptographic Step).

  To mitigate against this issue and claim FIPS compliance, RSA recommends customers either:

  - Upgrade to a higher, unaffected release - 6.2.5 at time of writing

    or

  - Apply the following workaround:

    - When using a `DHPublicKeySpec` to create a JCE public key, check that the DH public (Y) is non-negative. For example:

      ```
      BigInteger y = new BigInteger("1");
      if (y.compareTo(BigInteger.ZERO) < 0) {
          throw new RuntimeException("Negative DH public value");
      }
      BigInteger p = new BigInteger("1");
      BigInteger g = new BigInteger("1");
      DHPublicKeySpec dhPublicKeySpec = new DHPublicKeySpec(y, p, g);
      ```

    - Use the Assurance public API to validate the public key. For example:

      ```
      if (Assurance.isValidPublicKey(publicKey)) {
          Print.println("This Public key is valid according to the SP
          800-56A standard.");
      } else {
          throw new SampleFailedException("This Public key is not
          valid according to the SP 800-56A standard.");
      }
      ```

2.3.2 Crypto User Guidance on Obtaining Assurances for Digital Signature Applications

The module provides support for the FIPS 186-4 standard for digital signatures. The following gives an overview of the assurances required by FIPS 186-4. NIST Special Publication 800-89: “Recommendation for Obtaining Assurances for Digital Signature Applications” provides the methods to obtain these assurances.

The tables below describe the FIPS 186-4 requirements for signatories and verifiers and the corresponding module capabilities and recommendations.

Table 6 Signatory Requirements

FIPS 186-4 Requirement	Module Capabilities and Recommendations
Obtain appropriate DSA and ECDSA parameters when using DSA or ECDSA.	The generation of DSA parameters is in accordance with the FIPS 186-4 standard for the generation of probable primes. For ECDSA, use the NIST recommended curves as defined in section 2.3.1.
Obtain assurance of the validity of those parameters.	The module provides APIs to validate DSA parameters for probable primes as described in FIPS 186-4. For ECDSA, use the NIST recommended curves as defined in section 2.3.1. For the JCM API, `com.rsa.crypto.AlgParamGenerator.verify()`
Obtain a digital signature key pair that is generated as specified for the appropriate digital signature algorithm.	The module generates the digital signature key pair according to the required standards. Choose a FIPS-Approved RNG like HMAC DRBG to generate the key pair.
Obtain assurance of the validity of the public key.	The module provides APIs to explicitly validate the public key according to NIST Special Publication 800-89. For the JCM API, `com.rsa.crypto.PublicKey.isValid(com.rsa.crypto.SecureRandom secureRandom)`
Obtain assurance that the signatory actually possesses the associated private key.	The module verifies the signature created using the private key, but all other assurances are outside the scope of the module.

Table 7 Verifier Requirements

FIPS 186-4 Requirement	Module Capabilities and Recommendations
Obtain assurance of the signatory's claimed identity.	The module verifies the signature created using the private key, but all other assurances are outside the scope of the module.
Obtain assurance of the validity of the domain parameters for DSA and ECDSA.	The module provides APIs to validate DSA parameters for probable primes as described in FIPS 186-4. For the JCM API, `com.rsa.crypto.AlgParamGenerator.verify()` For ECDSA, use the NIST recommended curves as defined in section 2.3.1.
Obtain assurance of the validity of the public key.	The module provides APIs to explicitly validate the public key according to NIST Special Publication 800-89. For the JCM API, `com.rsa.crypto.PublicKey.isValid(com.rsa.crypto.SecureRandom secureRandom)`
Obtain assurance that the claimed signatory actually possessed the private key that was used to generate the digital signature at the time that the signature was generated.	Outside the scope of the module.

2.3.3 Crypto User Guidance on Key Generation and Entropy

The module generates cryptographic keys whose strengths are modified by available entropy. As a result, no assurance is given for the minimum strength of generated keys. The JCM provides the HMAC DRBG, CTR DRBG and Hash DRBG implementations for key generation.

When generating secure keys, the DRBG used in key generation must be seeded with a number of bits of entropy that is equal to or greater than the security strength of the key being generated. The entropy supplied to the DRBG is referred to as the DRBG security strength.

The following table lists each of the keys that can be generated by the JCM, with the key sizes available, security strengths for each key size and the security strength required to initialize the DRBG.

Table 8 Generated Key Sizes and Strength

Key Type	Key Size	Security Strength	Required DRBG Security Strength
AES Key	128, 192, 256	128, 192, 256	128, 192, 256
Triple-DES 3-Key	192	112	112
RSA Key Pair	2048, 3072	112, 128	112, 128

Table 8 Generated Key Sizes and Strength (continued)

Key Type	Key Size	Security Strength	Required DRBG Security Strength
DSA Key Pair	2048, 3072, 4096	112, 128, 128	112, 128, 128
EC Key Pair	224, 256, 384, 521	112, 128, 192, 256	112, 128, 192, 256

2.3.4 General Crypto User Guidance

JCM users should take care to zeroize CSPs when they are no longer needed. For more information on clearing sensitive data, see section 1.5.4 and the relevant API *Javadoc*.

2.4 Crypto Officer Guidance

The Crypto Officer is responsible for installing the module. Installation instructions are provided in the *RSA BSAFE Crypto-J Installation Guide*.

The Crypto Officer is also responsible for loading the module, as specified in section 2.1 Module Configuration.

2.5 Operating the Cryptographic Module

Both FIPS and non-FIPS algorithms are available to the operator. In order to operate the module in the FIPS-Approved mode, all rules and guidance provided in Secure Operation of the Module **must** be followed by the module operator. The module **does not** enforce the FIPS 140-2 mode of operation.

3 Acronyms

The following table lists the acronyms used with the JCM and their definitions.

Table 9 Acronyms used with the JCM

Acronym	Definition
3DES	Refer to Triple-DES
AD	Authenticated Decryption. A function that decrypts purported ciphertext and verifies the authenticity and integrity of the data.
AE	Authenticated Encryption. A block cipher mode of operation which provides a means for the authenticated decryption function to verify the authenticity and integrity of the data.
AES	Advanced Encryption Standard. A fast block cipher with a 128-bit block, and keys of lengths 128, 192 and 256 bits. This will replace DES as the US symmetric encryption standard.
API	Application Programming Interface.
Attack	Either a successful or unsuccessful attempt at breaking part or all of a crypto-system. Attack types include an algebraic attack, birthday attack, brute force attack, chosen ciphertext attack, chosen plaintext attack, differential cryptanalysis, known plaintext attack, linear cryptanalysis, middleperson attack and timing attack.
BPS	BPS is a format preserving encryption mode. BPS stands for Brier, Peyrin and Stern, the inventors of this mode.
CBC	Cipher Block Chaining. A mode of encryption in which each ciphertext depends upon all previous ciphertexts. Changing the IV alters the ciphertext produced by successive encryptions of an identical plaintext.
CFB	Cipher Feedback. A mode of encryption that produces a stream of ciphertext bits rather than a succession of blocks. In other respects, it has similar properties to the CBC mode of operation.
CRNG	Continuous Random Number Generation.
CSP	Critical Security Parameters.
CTR	Counter mode of encryption, which turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a counter.
CTS	Cipher Text Stealing. A mode of encryption which enables block ciphers to be used to process data not evenly divisible into blocks, without the length of the ciphertext increasing.
DES	Data Encryption Standard. A symmetric encryption algorithm with a 56-bit key.

Table 9 Acronyms used with the JCM (continued)

Acronym	Definition
Diffie-Hellman	The Diffie-Hellman asymmetric key exchange algorithm. There are many variants, but typically two entities exchange some public information (for example, public keys or random values) and combines them with their own private keys to generate a shared session key. As private keys are not transmitted, eavesdroppers are not privy to all of the information that composes the session key.
DPK	Data Protection Key.
DRBG	Deterministic Random Bit Generator.
DSA	Digital Signature Algorithm. An asymmetric algorithm for creating digital signatures.
EC	Elliptic Curve.
ECB	Electronic Code Book. A mode of encryption in which identical plaintexts are encrypted to identical ciphertexts, given the same key.
ECC	Elliptic Curve Cryptography.
ECDH	Elliptic Curve Diffie-Hellman.
ECDSA	Elliptic Curve Digital Signature Algorithm.
ECIES	Elliptic Curve Integrated Encryption Scheme.
Encryption	The transformation of plaintext into an apparently less readable form (called ciphertext) through a mathematical process. The ciphertext may be read by anyone who has the key that decrypts (undoes the encryption) the ciphertext.
FIPS	Federal Information Processing Standards.
FPE	Format Preserving Encryption.
HMAC	Keyed-Hashing for Message Authentication Code.
IV	Initialization Vector. Used as a seed value for an encryption operation.
JCE	Java Cryptography Extension.
JVM	Java Virtual Machine.
KAT	Known Answer Test.
KDF	Key Derivation Function. Derives one or more secret keys from a secret value, such as a master key, using a pseudo-random function.

Table 9 Acronyms used with the JCM (continued)

Acronym	Definition
Key	A string of bits used in cryptography, allowing people to encrypt and decrypt data. Can be used to perform other mathematical operations as well. Given a cipher, a key determines the mapping of the plaintext to the ciphertext. Various types of keys include: distributed key, private key, public key, secret key, session key, shared key, subkey, symmetric key, and weak key.
KW	AES Key Wrap.
KWP	AES Key Wrap with Padding
MD5	A secure hash algorithm created by Ron Rivest. MD5 hashes an arbitrary-length input into a 16-byte digest.
NIST	National Institute of Standards and Technology. A division of the US Department of Commerce (formerly known as the NBS) which produces security and cryptography-related standards.
OFB	Output Feedback. A mode of encryption in which the cipher is decoupled from its ciphertext.
OS	Operating System.
PBE	Password-Based Encryption.
PBKDF	Password-Based Key Derivation Function.
PC	Personal Computer.
private key	The secret key in public key cryptography. Primarily used for decryption but also used for encryption with digital signatures.
PRNG	Pseudo-random Number Generator.
RC2	Block cipher developed by Ron Rivest as an alternative to the DES. It has a block size of 64 bits and a variable key size. It is a legacy cipher and RC5 should be used in preference.
RC4	Symmetric algorithm designed by Ron Rivest using variable length keys (usually 40 bit or 128 bit).
RC5	Block cipher designed by Ron Rivest. It is parameterizable in its word size, key length and number of rounds. Typical use involves a block size of 64 bits, a key size of 128 bits and either 16 or 20 iterations of its round function.
RNG	Random Number Generator.
RSA	Public key (asymmetric) algorithm providing the ability to encrypt data and create and verify digital signatures. RSA stands for Rivest, Shamir, and Adleman, the developers of the RSA public key cryptosystem.

Table 9 Acronyms used with the JCM (continued)

Acronym	Definition
SHA	Secure Hash Algorithm. An algorithm which creates a hash value for each possible input. SHA takes an arbitrary input which is hashed into a 160-bit digest.
SHA-1	A revision to SHA to correct a weakness. It produces 160-bit digests. SHA-1 takes an arbitrary input which is hashed into a 20-byte digest.
SHA-2	The NIST-mandated successor to SHA-1, to complement the Advanced Encryption Standard. It is a family of hash algorithms (SHA-256, SHA-384 and SHA-512) which produce digests of 256, 384 and 512 bits respectively.
Shamir Secret Sharing	A form of secret sharing where a secret is divided into parts, and each participant is given a unique part. Some or all of the parts are needed to reconstruct the secret. This is also known as a (k,n) threshold scheme where any k of the n parts are sufficient to reconstruct the original secret.
TDES	Refer to Triple-DES.
TLS	Transport Layer Security.
Triple-DES	A symmetric encryption algorithm which uses either two or three DES keys. The two key variant of the algorithm provides 80 bits of security strength while the three key variant provides 112 bits of security strength.

4 Change Summary

11 March 2020 Addition - Section 2.3.1 Crypto User Guidance on Algorithms
The JCM is affected by CVE-2019-3738 (Missing Required Cryptographic Step).

24 August 2020 Update - Preface - Notice of Dell acquisition
Update - Section 2.1 Module Configuration - path change.

20 October 2020Update Section 1.4 Roles and Services - removal of Level 2 text.