RSA® BSAFE® Crypto-J JSAFE and JCE Software Module 6.2.4 Security Policy Level 1

This document is a non-proprietary security policy for the RSA BSAFE Crypto-J JSAFE and JCE Software Module 6.2.4 (Crypto-J JSAFE and JCE Software Module) security software.

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 Technologies 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.

1.2.1 Laboratory Validated Operating Environments

For FIPS 140-2 validation, the JCM is tested by an accredited FIPS 140-2 testing laboratory on the following operating environments:

- Google™ ART™ JRE 8.0 on Google Android™ 7.1.2 ARMv8 (64-bit) running on Google Nexus™ 5x with a Qualcomm® Snapdragon™ processor
- Oracle® JRE 8.0 on Microsoft® Windows® 10 (64-bit) running on Dell™ OptiPlex™ with an Intel® Core™ i5 processor.
- OpenJDK 8.0 on CentOS 7.3 (64-bit) running on Dell PowerEdge™ with an Intel Xeon® processor.

1.2.2 Affirmation of Compliance for other Operating Environments

Affirmation of compliance is defined in Section G.5, “Maintaining Validation Compliance of Software or Firmware Cryptographic Modules,” in Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program. Compliance is maintained in all operational environments for which the binary executable remains unchanged. Specifically, Dell Technologies affirms compliance for the following operational environments:

- Apple®
  - Mac OS® X 10.6+
    - x86 (32-bit) Apple JDK 8.0
    - x86_64 (64-bit) Apple JDK 8.0.
- Canonical™
  - Ubuntu™ 16.04 Server
    - x86 (32-bit) IBM JDK 8.0, OpenJDK 8u, Oracle JDK 8.0, Oracle JRockit 6.0
    - x86_64 (64-bit) IBM JDK 8.0, OpenJDK 8u, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.
- Google®
  - Android™ 4.4.x
    - ARM® v7 (32-bit) Android JDK 7.0
    - x86 (32-bit) Android JDK 7.0.
  - Android 5.x
    - ARM v7 (32-bit) Android JDK 7.0
    - x86 (32-bit) Android JDK 7.0.
  - Android 6.x
    - ARM v7 (32-bit) Android JDK 7.0
    - ARM v8 (32-bit) Android JDK 7.0
    - ARM v8 (64-bit) Android JDK 7.0
    - x86 (32-bit) Android JDK 7.0.
  - Android 7.x
    - ARM v7 (32-bit) Android JDK 7.0
    - ARM v8 (32-bit) Android JDK 7.0
    - ARM v8 (64-bit) Android JDK 7.0
    - x86 (32-bit) Android JDK 7.0.

  - Android 8.0
    - ARM v7 (32-bit) Android JDK 7.0
    - ARM v8 (32-bit) Android JDK 7.0
    - ARM v8 (64-bit) Android JDK 7.0
    - x86 (32-bit) Android JDK 7.0.
- HP
  - HP-UX 11.31
    - Itanium® 2 (32-bit) HP JDK 8.0
    - Itanium 2 (64-bit) HP JDK 8.0.
- IBM
  - AIX® 7.1
    - PowerPC® (32-bit) IBM JDK 8.0
    - PowerPC (64-bit) IBM JDK 8.0.
  - AIX 7.2
    - PowerPC (32-bit) IBM JDK 8.0
    - PowerPC (64-bit) IBM JDK 8.0.
- Linux®
  - CentOS 6.9
    - x86_64 (64-bit) IBM JDK 8.0, OpenJDK 8.u, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.
  - CentOS 7.4
    - x86_64 (64-bit) IBM JDK 8.0, OpenJDK 8.u, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.
  - Micro Focus® SUSE Linux Enterprise Server 12.0 SP2
    - x86_64 (64-bit) IBM JDK 8.0 OpenJDK 8u, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.
  - Red Hat® Enterprise Linux 7.3
    - x86_64 (64-bit) IBM JDK 8.0, OpenJDK 8.u, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.
- Microsoft®
  - Windows® 7 Enterprise SP1
    - x86 (32-bit) Oracle JRockit 6.0
    - x86_64 (64-bit) IBM JDK 8.0, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.

  - Windows 8.1 Enterprise
    - x86 (32-bit) Oracle JRockit 6.0
    - x86_64 (64-bit) IBM JDK 8.0, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.
  - Windows 10 Enterprise
    - x86 _64 (64-bit) IBM JDK 8.0, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.
  - Windows Server 2008 SP2
    - x86 (32-bit) Oracle JRockit 6.0
    - x86_64 (64-bit) IBM JDK. 8.0, Oracle JDK 8.0, Oracle JRockit 6.0.
  - Windows Server 2012
    - x86_64 (64-bit) IBM JDK 8.0, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.
  - Windows Server 2012 R2
    - x86_64 (64-bit) IBM JDK 8.0, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.
  - Windows Server 2016
    - x86_64 (64-bit) IBM JDK 8.0, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0.
- Oracle
  - Solaris® 10
    - SPARC® v9 (64-bit) IBM JDK 8.0, Oracle JDK 8.0, Oracle JRockit 6.0
    - x86_64 (64-bit) Oracle JDK 8.0.
  - Solaris 11
    - SPARC v9 (64-bit) IBM JDK 8.0, Oracle JDK 8.0 and 9.0.1, Oracle JRockit 6.0
    - x86_64 (64-bit) Oracle JDK 8.0.
- The FreeBSD® Foundation
  - FreeBSD 10.3
    - x86_64 (64-bit) OpenJDK 8.u.
  - FreeBSD 11.x
    - x86_64 (64-bit) OpenJDK 8.u.

Note: The Cryptographic Module Validation Program (CMVP) makes no statement as to the correct operation of the module or the security strengths of the generated keys when the specific operational environment is not listed on the validation certificate.

1.3 Module Interfaces

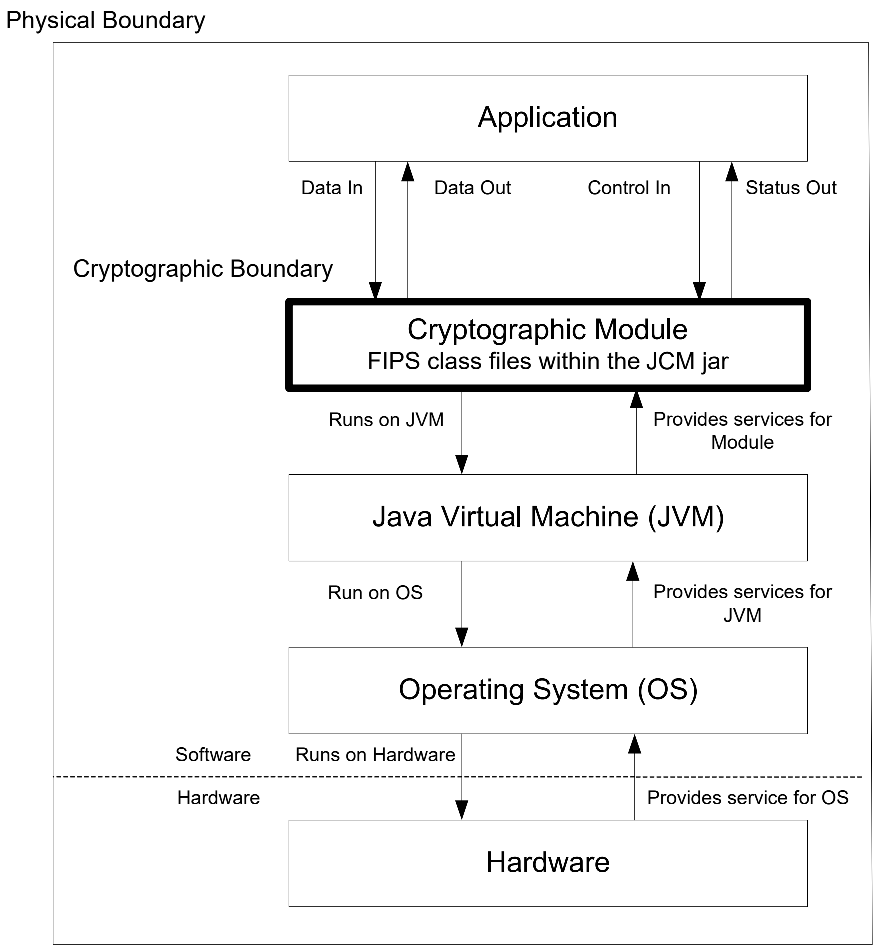
As a multi-chip standalone module, the physical interface to the JCM consists of a keyboard, mouse, monitor, serial ports and network adapters.

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 .

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 (CBC, CCM, CFB, CTR, ECB, GCM, OFB, XTS) Triple-DES (CBC, CFB, ECB, 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, provides 112 or 128 bits of encryption strength) SP800-38F KW (AE, AD, provides between 128 and 256 bits of encryption strength) SP800-38F KWP (AE, AD, provides between 128 and 256 bits of encryption strength) RSA-KEM-KWS (When used with approved KDF and Key Wrap algorithms)	
	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`
`PQGParams`	`getCryptoModule` `getG`	`getP` `getQ`
Parameter Generation:		

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

	Algorithms allowed for FIPS 140-2 usage	
	EC DSA Diffie-Hellman (DH)	
	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 Establishment:		

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

	Algorithms allowed for FIPS 140-2 usage	
	Diffie-Hellman (primitives only, provides 112 bits or 128 bits of encryption strength) EC Diffie-Hellman (primitives only, provides between 112 bits and 256 bits of encryption strength)	
	Algorithms not allowed for FIPS 140-2 usage	
	None	
`KeyConfirmation`	`clearSensitiveData` `computeMacTag`	`verifyMacTag`
	Algorithms allowed for FIPS 140-2 usage	
	Diffie-Hellman (primitives only, provides 112 bits or 128 bits of encryption strength) EC Diffie-Hellman (primitives only, provides between 112 bits and 256 bits of encryption strength)	
	Algorithms not allowed for FIPS 140-2 usage	
	None	
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 Shamir Secret Sharing	RC2 RC4 RC5

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: (continued)		

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

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

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

`DHPublicKey`	`clearSensitiveData`
	`clone`
	`getAlg`
	`getCryptoModule`
		`getParams`
		`getY`
		`isValid`

`DSAPrivateKey`	`clearSensitiveData`
	`clone`
	`getAlg`
	`getCryptoModule`
		`getParams`
		`getX`
		`isValid`

`DSAPublicKey`	`clearSensitiveData`
	`clone`
	`getAlg`
	`getCryptoModule`
		`getParams`
		`getY`
		`isValid`

`ECPrivateKey`	`clearSensitiveData`
	`clone`
	`getAlg`
	`getCryptoModule`
		`getD`
		`getParams`
		`isValid`

`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 isValid
PublicKey	clearSensitiveData clone getAlg	getCryptoModule isValid
RSAPrivateKey	clearSensitiveData clone getAlg getCoeff getCryptoModule getD getE getExpP	getExpQ getN getOtherMultiPrimeInfo getP getQ hasCRTInfo isMultiprime isValid
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) Single-step KDF	
	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`

`CryptoModule`	`getDeviceType`
	`getKeyBuilder`
	`getModuleOperations`
	`newAlgInputParams`
	`newAlgParamGenerator`
	`newAsymmetricCipher`
	`newKDF`
	`newKeyAgreement`
		`newKeyGenerator`
		`newKeyPairGenerator`
		`newKeyWrapCipher`
		`newMAC`
		`newMessageDigest`
		`newSecureRandom`
		`newSignature`
		`newSymmetricCipher`

`JCMCloneable`	`clone`	

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)		

`ModuleConfig`	`getEntropySource`
	`getFIPS140Context`
	`getSecurityLevel`
	`getVersionDouble`
	`getVersionInfo`
	`getVersionString`
		`initFIPS140RolePINs`
		`isFIPS140Compliant`
		`newCryptoModule`
		`runSelfTests`
		`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

Cryptographic key management is concerned with generating and storing keys, protecting keys during use, zeroizing keys when they are no longer required and managing access to keys.

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 JCM 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)

Table 3 Key and CSP Storage (continued)

Item	Storage
ECC private keys/public key	In volatile memory only (plaintext)
ECDH Shared Secret	In volatile memory only (plaintext)
DH Shared Secret	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)
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)
Katstatus	In volatile memory and on disk (protected with HMAC SHA256)

1.5.5 Key Access

An authorized operator of the module has access to all key data created during JCM operation. 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, ECDSA, RSA)	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 Single-step KDF keys	Read/Execute
Key establishment	Asymmetric keys (DH, ECDH)	Read/Execute
Key generation	Symmetric keys (AES, Triple-DES) Asymmetric keys (DH, DSA, ECDSA, ECDH, RSA) MAC keys (HMAC)	Write
Self-test	Hard-coded keys, (AES, Triple-DES, RSA, DSA, ECDSA, HMAC) Hard-coded entropy, strength, and seed (HMAC DRBG, HASH DRBG, CTR 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	Standard	Validation Certificate
Asymmetric Cipher	RSA-OAEP, RSA-KEM-KWS (2048 and 3072 bit key sizes)		Vendor Affirmed as part of Key Transport Schemes
Asymmetric Key	RSADP Component Test		Certificate #1572
Key Agreement Primitives	FFC DH (2048 and 3072 bit key sizes)	SP 800-56A	Vendor affirmed
	ECDHC (224 to 571 bit key sizes)		
	KASECC_(ECCCDH) Primitive Component Test		Certificate #1570
Key Agreement Schemes	FFC, ECC primitive / Single-Step KDF / Key Confirmation [dhHybrid1, dhEphem, dhHybridOneFlow, dhOneFlow, dhStatic, (Cofactor) Full Unified Model, (Cofactor) Ephemeral Unified Model, (Cofactor) One-Pass Unified Model, (Cofactor) One-Pass Diffie-Hellman, (Cofactor) Static Unified Model]		Certificate #156
Key Transport Schemes	RSA-OAEP, RSA-KEM-KWS cipher / Single-Step KDF / Key Confirmation [KTS-OAEP, KTS-OAEP-Party_V-confirmation, KTS-KEM-KWS, KTS-KEM-KWS-Party_V-confirmation]	SP 800-56B	Vendor affirmed
Key Derivation	PBKDF2	SP 800-132	Vendor Affirmed (Approved in FIPS mode for key storage*)
	KDFTLS10**	SP 800-135 rev1	Certificate #1571
	KDFTLS12*** with SHA-256, SHA-384, SHA-512		
Key Wrap	KTS (AES Certificate #5026: key establishment methodology provides between 128 and 256 bits of encryption strength)	SP800-38F	
Key Generation	Cryptographic Key Generation (CKG)	SP800-133	Vendor affirmed
Message Authentication Code	HMAC****	FIPS 198-1	Certificate #3340

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

Algorithm Type	Algorithm	Standard	Validation Certificate
Message Digest	SHA-1[*****], SHA-224, SHA-256, SHA-384, SHA-512	FIPS 180-4	Certificate #4085
	SHA-512/224, SHA-512/256		
Random Bit Generator	CTR DRBG	SP 800-90A rev1	Certificate #1841
	Hash DRBG		
	HMAC DRBG	SP 800-90A rev1	
Signature[*****]	RSA X9.31, PKCS #1 V.1.5, RSASSA-PSS (2048 and 3072 bit key sizes)		Certificate #2711
	DSA (2048 and 3072 bit key sizes)	FIPS 186-4	Certificate #1321
	ECDSA (224 to 571 bit key sizes)		Certificate #1283
Symmetric Cipher	AES in CBC, CFB, CTR, ECB, OFB modes (128, 192, 256 bit key sizes)	SP 800-38A	Certificate #5026
	AES in CCM modes (128, 192, 256 bit key sizes)	SP 800-38C	
	AES in GCM mode (128, 192, 256 bit key sizes)	SP 800-38D	
	AES in XTS mode (128, 256 bit key sizes)	SP 800-38E	
	Triple-DES[******] (CBC, CFB, ECB, OFB)	SP 800-67 and SP 800-38A	Certificate #2591

[*]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.

[*****]SHA-1 is allowed for use in the TLS protocol but no longer allowed to be used in digital signature generation.

[******]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

Power-up self-tests are executed automatically when the module is loaded into memory. The power-up self-tests include the FIPS140-2 required Software Integrity Test and a set of Cryptographic Algorithms tests. The following Cryptographic Algorithm tests are implemented in the module:

- AES Decrypt KAT
- AES Encrypt KAT
- CTR DRBG KAT
- Diffie-Hellman KAT
- DSA Sign/Verify Test
- EC Diffie-Hellman KAT
- ECDSA Sign/Verify Test
- Hash DRBG KAT
- HMAC DRBG KAT
- KDFTLS10 KAT
- KDFTLS12 SHA-256 KAT
- RSA Signature Generation KAT
- RSA Signature Verification KAT
- SHA-512 KAT
- Triple-DES Decrypt KAT
- Triple-DES Encrypt KAT.

By default, all Cryptographic Algorithm tests are run at power-up. However, if configured to do so, the module will run all of the power-up self-tests when first loaded in an operational environment, and run only the Software Integrity Test on subsequent restarts.

1.7.2 Conditional Self-tests

The module performs two conditional self-tests:

- Pair-wise Consistency Tests each time the module generates a DSA, DH, 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.

For other types of timing attacks the module implements time invariant comparisons and operations (for example, PKCS #1 unpadding, HMAC verify, and RSA verify).

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.jcm.module.CryptoJModulePropertiesFactory` class.

The following arguments are extracted:

- The module jar file.
- The security level, specified as the constant `ModuleConfig.LEVEL_1`. This should have a value of 1.
- An optional `SelfTestEventListener` to use for logging power-up self-test events.
- An optional `java.util.concurrent.ExecutorService` used for running the power-up self-tests.
- An optional `File` for reading and writing the status of the algorithm power-up self-tests.

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 obtaining assurances for Key Agreement Applications.

Section 2.3.4 provides guidance on obtaining assurances for Key Transport Applications.

Section 2.3.5 provides guidance on the entropy requirements for secure key generation.

Section 2.3.6 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.
- Only FIPS 140-2 Approved DRBGs may be used for generation of keys (asymmetric and symmetric).
- When using an approved DRBG, 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 DRBG to generate keys or DSA parameters, the DRBG's requested security strength must be at least as great as the security strength of the key being generated. That means that an Approved DRBG with an appropriate strength must be used. For more information on requesting the DRBG security strength, see the relevant API *Javadoc*.
- Since the module does not modify the output of an Approved DRBG, any generated symmetric keys or seed values are created directly from the output of the Approved DRBG.
- 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.
- The AES-GCM cipher, when used for symmetric encryption purposes other than TLS, must use an IV in one of the two possible ways, to comply with SP 800-38D:
  - allow the module to generate the IV deterministically by not supplying any IV parameters during cipher initialization. The generated 96-bit (12-byte) IV consists of a 32-bit fixed field followed by a 64-bit invocation field where
    - the fixed field bytes are derived from the module name, version information and memory address of a Java class within the module
    - the invocation field is a 64-bit counter that is initialized, on startup, to a value consisting of the 44 bits of current time, as milliseconds since Epoch, followed by 22 bits of zero. By using the current time to prefix the counter start value, in the event of module restart, the counter will be ahead of any previous module states, ensuring that IV values cannot be reused.
  - generate at least 12 bytes of IV using an Approved DRBG, and input the IV to the cipher at initialization time using the `RAW_IV` parameter.
- The AES-GCM cipher used for the TLS protocol as the cipher implementation complies with SP 800-52 and is compatible with RFC 5288 if the IV is configured as follows:

  The four-byte *salt* derived from the TLS handshake process is input using the parameter `PARTIAL_IV` during cipher initialization. This is used as the first four bytes of IV. The remaining eight bytes of IV, referred to as *nonce_explicit* in RFC 5288, are generated deterministically by the module using the 64-bit counter used for the invocation field described above.
- 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.
- When using DSA key pair generation and signature generation or DH key pair generation and key agreement, the domain parameters must have prime P length (`PRIME_LEN`) and subprime Q length (`SUBPRIME_LEN`) within the set of acceptable pair sets (`PRIME_LEN, SUBPRIME_LEN`): (2048, 224), (2048, 256) or (3072, 256).
- When generating DSA and DH domain parameters, generation shall comply with FIPS 186-4 by specifying the algorithm string "DSA" when creating the `AlgParameterGenerator` object. Additionally:
  - The `PRIME_LEN` and `SUBPRIME_LEN` must be from a set of acceptable pair set as stated above.
  - The digest algorithm parameter shall be selected from the set: SHA224, SHA256, SHA384, SHA512. The digest algorithm shall be selected such that the output length is at least as large as the subprime length. That is:
    - For subprime 224, all four algorithms may be used.
    - For subprime 256, only SHA256, SHA384, SHA512 may be used.
- For RSA digital signature generation, an Approved DRBG must be used.
- RSA keys used for signing shall not be used for any other purpose other than digital signatures.
- When generating RSA key pairs for signatures or key transport, generation shall comply with the following:
  - the `KEY_TYPE` parameter must be omitted or have a value of 0.
  - the `KEY_BITS` parameter must have value 2048 or 3072.
  - the `SECURITY_STRENGTH` parameter may be input. Acceptable values are:
    - 112, when used for `KEY_BITS` of 2048.
    - 128, when used for `KEY_BITS` of 3072.
  - the `PUB_EXP` value must be an odd number and have a minimum value of 0x10001 (65537).
- 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 equal to or greater than 112 bits. For HMAC verification, a key length greater than or equal to 80 and less than 112 is allowed for legacy-use.

Note: JCE MAC APIs do not distinguish between generate and verify, therefore a key length check is not explicitly performed in JCE.

- When using Single-step KDF, a FIPS 140-2 approved hash function must be used.
- 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.
  - The use of three-key Triple-DES is limited to 2^{32} encryption operations when the key is generated as a part of one of the following IETF protocols: TLS 1.0 as specified in RFC 2246, TLS 1.1 as specified in RFC 4346, or TLS 1.2 as specified in RFC 5246.
  - The use of three-key Triple-DES is limited to 2^{28} encryption operations when the key is not generated as a part of a recognized IETF protocol.

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, `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 DRBG 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 SP 800-89. For the JCM API, `PublicKey.isValid(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, `AlgParamGenerator.verify()` For ECDSA, use the NIST recommended curves as defined in section 2.3.1.

Table 7 Verifier Requirements (continued)

FIPS 186-4 Requirement	Module Capabilities and Recommendations
Obtain assurance of the validity of the public key.	The module provides APIs to explicitly validate the public key according to SP 800-89. For the JCM API, `PublicKey.isValid(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 Obtaining Assurances for Key Agreement Applications

The module provides support for the NIST SP800.56A recommendations for key agreement. NIST Special Publication 800-56A: "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography" provides the methods to obtain these assurances.

The tables below describe the SP800-56A recommendations for key establishment and the corresponding module capabilities and recommendations.

Table 8 Key Establishment Recommendations

NIST SP800-56A Recommendations	Module Capabilities and Recommendations
Obtain appropriate FFC and ECC domain parameters.	The generation of FFC parameters is in accordance with the FIPS 186-4 standard for the generation of probable primes. For ECC, use the NIST recommended curves as defined in section 2.3.1.
Obtain assurance of the validity of those domain parameters.	The module provides APIs to validate FFC parameters for probable primes as described in FIPS 186-4. For ECC, use the NIST recommended curves as defined in section 2.3.1. For the JCM API, `AlgParamGenerator.verify()`
Obtain a key establishment key pair that is generated as specified for the appropriate algorithm.	The module generates the digital signature key pair according to the required standards. Choose a FIPS-Approved DRBG like HMAC DRBG to generate the key pair.
Owner assurance of the validity of the public key.	The module provides APIs to explicitly validate the public key according to SP 800-89. For the JCM API, `PublicKey.isValid(SecureRandom secureRandom)`

Table 8 Key Establishment Recommendations (continued)

NIST SP800-56A Recommendations	Module Capabilities and Recommendations
Owner assurance of the validity of the private key.	The module provides APIs to explicitly validate the private key according to SP 800-56A. For the JCM API, `PrivateKey.isValid()`
Owner assurance of pairwise consistency	The module provides an API to explicitly validate the keypair according to the pairwise consistency requirements in SP 800.56A. For the JCM API, `KeyPair.validate(SecureRandom)`

2.3.4 Crypto User Guidance on Obtaining Assurances for Key Transport Applications

The module provides support for the NIST SP800.56B recommendations for key transport. NIST Special Publication 800-56B Revision 1: "Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography" provides the methods to obtain these assurances.

The tables below describe the SP800-56B recommendations for key transport.

Table 9 Key Transport Recommendations

NIST SP800-56B Recommendations	Module Capabilities and Recommendations
Assurance of Key-Pair Validity	The module provides APIs to explicitly validate an RSA Key Pair according to SP 800.56B. The JCM API available is: `KeyPair.validate(AlgInputParams, SecureRandom).` The parameters object can be supplied with `SECURITY_STRENGTH` and `KEY_BITS` inputs. This API performs both a pairwise consistency test and a key pair validation according to "`rsakpv1-crt`" and "`crt_pkv`" methods.
Assurance of Public Key Validity	The module provides APIs to explicitly validate the RSA public key according to SP 800.56B and SP 800-89. The JCM API available is: `KeyPair.validate(AlgorithmParams, SecureRandom)`
Assurance of Possession of Private Key	The module supports Key Confirmation for providing assurance of possession of a private key in a key transport scheme. The JCM API available is: `KeyConfirmation`.

2.3.5 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 10 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
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

2.3.6 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 1.5.3 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 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 11 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.
CKG	Cryptographic Key Generation.
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.

Table 11 Acronyms used with the JCM (continued)

Acronym	Definition
DES	Data Encryption Standard. A symmetric encryption algorithm with a 56-bit key.
DH, 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.
ECDHC	Elliptic Curve Cryptography 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.
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standards.
FPE	Format Preserving Encryption.
HKDF	HMAC-based Extract-and-Expand Key Derivation Function.
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.

Table 11 Acronyms used with the JCM (continued)

Acronym	Definition
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.
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.

Table 11 Acronyms used with the JCM (continued)

Acronym	Definition
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 crypto-system.
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.