29.04.16

RSA BSAFE® Crypto-J JSAFE and JCE Software Module 6.1 Security Policy Level 1

This document is a non-proprietary security policy for RSA BSAFE Crypto-J JSAFE and JCE Software Module 6.1 and 6.1.1.0.1 (RSA BSAFE Crypto-J JSAFE and JCE Software Module) security software.

Contents:

Preface

This document is a non-proprietary security policy for the RSA BSAFE Crypto-J JSAFE and JCE Software Module from RSA, the Security Division of EMC (RSA).

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

FIPS 140-2 (Federal Information Processing Standards Publication 140-2 - Security Requirements for Cryptographic Modules) 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.

References

This document deals only with operations and capabilities of the RSA BSAFE Crypto-J JSAFE and JCE Software Module in the technical terms of a FIPS 140-2 cryptographic module security policy. More information on RSA BSAFE Crypto-J JSAFE and JCE Software Module and the entire RSA BSAFE product line is available at:

- http://www.rsa.com/, for information on the full line of products and services.
- http://www.rsa.com/node.aspx?id=1319 for an overview of security tools for Java developers.
- http://www.rsa.com/node.aspx?id=1204 for an overview of the RSA BSAFE product range.

Terminology

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

The *RSA BSAFE 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 RSA BSAFE Crypto-J JSAFE and JCE Software Module features and functionality relevant to FIPS 140-2, and contains the following sections:

- This section, "Preface" on page 2 provides an overview and introduction to the Security Policy.
- "The Cryptographic Module" on page 4, describes the module and how it meets the FIPS 140-2 Security Level 1 requirements.
- "Secure Operation of the Module" on page 24, addresses the required configuration for the FIPS 140-2 mode of operation.
- "Acronyms" on page 31, 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*, the FIPS 140-2 Security Level 1 Validation Submission Documentation is EMC Corporation-proprietary and is releasable only under appropriate non-disclosure agreements. For access to the documentation, please contact RSA.

1 The Cryptographic Module

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

1.1 Introduction

More than a billion copies of the RSA 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, RSA BSAFE software is a set of complementary security products relied on by developers and manufacturers worldwide.

The Crypto-J software library relies on the Java Cryptographic Module 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 SHA1 and SHA2 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

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

JCM is validated for FIPS 140-2 Security Level 1 requirements. JCM is packaged in a Java Archive (JAR) file containing all the code for the module.

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

JCM is tested on the following platforms:

- Oracle® JRE 7.0 on Microsoft® Windows 7™ (64-bit) running on Dell™ Dimension C521
- JRE 6.0 on Google™ Android™ 2.2 ARM (32-bit) running on Lenovo® Think Pad® T61 (single-user mode).

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
    - x86_64 (64-bit), Apple JDK 6.0.
- Canonical™
  - Ubuntu™ 10.04
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, Oracle JRockit® 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0.
- Google
  - Android 2.1 ARM (32-bit) JDK 6.0
  - Android 2.2 ARM (32-bit) JDK 6.0
  - Android 2.3 ARM (32-bit) JDK 6.0
  - Android 4.0 ARM (32-bit) JDK 6.0.
- HP
  - HP-UX 11.31
    - PA-RISC 2.0 (32-bit), HP JRE 6.0
    - PA-RISC 2.0W (64-bit), HP JRE 6.0
    - Itanium2 (32-bit), HP JRE 6.0/7.0
    - Itanium2 (64-bit), HP JRE 6.0/7.0.
  - OpenVMS 8.3-1H1
    - Itanium2 (64-bit), HP JRE 6.0.
  - OpenVMS 8.4
    - Itanium2 (64-bit), HP JRE 6.0.

- 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®
  - Novell® SUSE® 10
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0.
  - Novell SUSE 11
    - Itanium 64-bit, Oracle JRE 6.0.
  - Novell SUSE Linux Enterprise Server 10
    - PowerPC (32-bit), IBM JRE 6.0/7.0
    - PowerPC (64-bit), IBM JRE 6.0/7.0.
  - Novell SUSE Linux Enterprise 11
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0.
  - Novell SUSE Linux Enterprise Server 11
    - PowerPC (32-bit), IBM JRE 6.0/7.0
    - PowerPC (64-bit), IBM JRE 6.0/7.0.
  - Red Hat® Enterprise Linux AS 5.0
    - PowerPC (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0
    - PowerPC (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0.
  - Red Hat Enterprise Linux 5.5, Security Enhanced Linux Configuration
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0.
  - Red Hat Enterprise Linux 6.0
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0.

  - Red Hat Enterprise Server 5.5
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0/7.0, JRockit 6.0.
- Microsoft
  - Windows XP Professional SP3
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0.
  - Windows Server 2003
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - Itanium (64-bit), Oracle JRE 6.0.
  - Windows Server 2008
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - Itanium (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0.
  - Windows Server 2008 (SSLF configuration)
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0.
  - Windows Vista® (SSLF configuration)
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0.
  - Windows Vista Ultimate
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0.
  - Windows 7
    - x86 (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0.
- Oracle
  - Solaris™ 10
    - SPARC v8+ (32-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - SPARC v9 (64-bit), Oracle JRE 6.0/7.0, IBM JRE 6.0, JRockit 6.0
    - x86 (32-bit), Oracle JRE 6.0/7.0
    - x86_64 (64-bit), Oracle JRE 6.0/7.0.

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 for Control Input through the API calls. Data Input and Output are provided in the variables passed with API calls, and Status Output is provided in the returns and error codes documented for each call. This is shown in the following diagram.

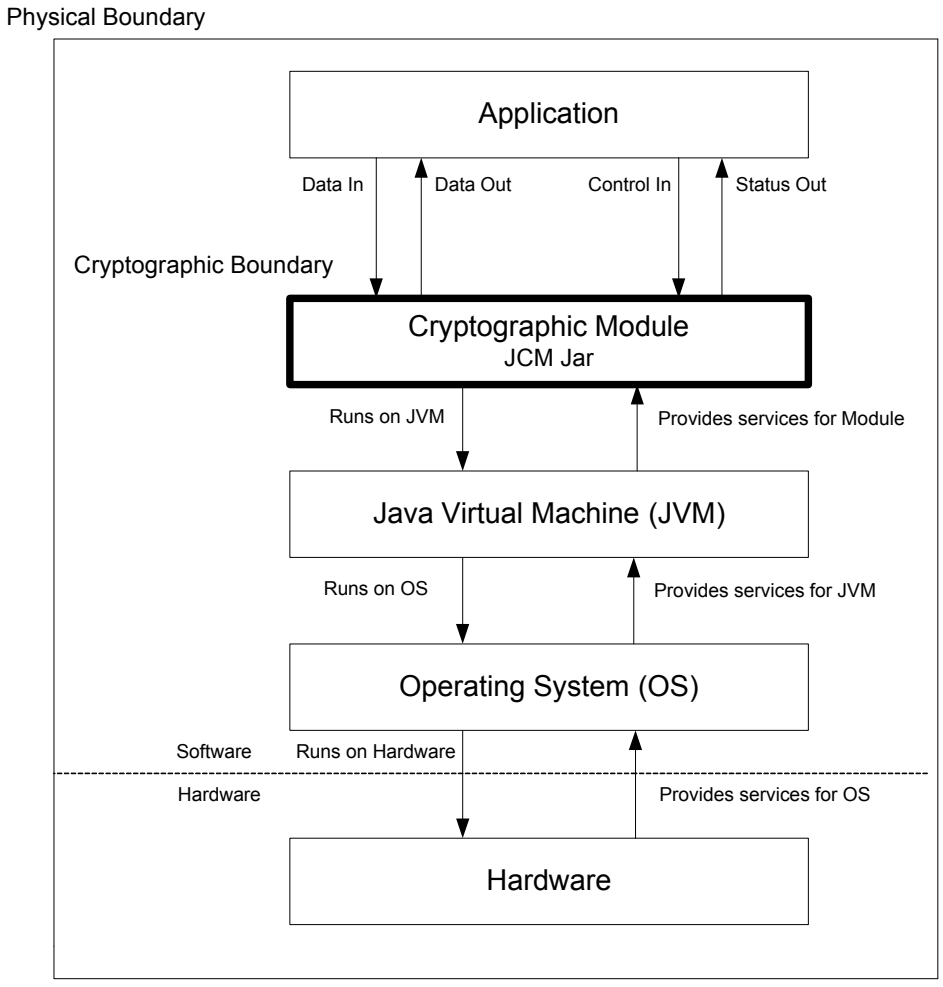

Figure 1 JCM Logical Diagram

1.4 Roles and Services

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, JCM does not require user identification or authentication for these roles.

The API for control of the JCM is through the `com.rsa.crypto.ModuleConfig` class.

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 in **both** FIPS and non-FIPS mode. In non-FIPS mode, the module is able to accept both FIPS 140-2 approved and Non-Approved algorithms. In FIPS mode, the module enforces the use of FIPS 140-2 approved algorithms. For a list of the FIPS 140-2 approved algorithms, see Table 4 on page 21.

The following table lists the un-authenticated services provided by JCM which may be used by either Role, in either the FIPS or non-FIPS mode, in terms of the module interface.

Table 1 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
Cipher	clearSensitiveData clone doFinal getAlg getAlgorithmParams getBlockSize getCryptoModule getMaxInputLen getOutputSize init reInit update
Signature Generation/Verification:	
Signature	clearSensitiveData clone getAlg getCryptoModule getSignatureSize initSign initVerify reInit sign update verify

Table 1 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
Digest Generation:	
MessageDigest	clearSensitiveData clone digest getAlg getCryptoModule getDigestSize reset update
Key Establishment Primitives:	
KeyAgreement	clearSensitiveData clone doPhase getAlg getCryptoModule getSecret init
Parameters:	
AlgInputParams	clone get getCryptoModule set
AlgorithmParams	getCryptoModule
DHParams	getCounter getCryptoModule getG getJ getMaxExponentLen getP getQ getSeed
DomainParams	getCryptoModule

Table 1 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Parameters: (continued)	
DSAParams	getCounter getCryptoModule getDigestAlg getG getP getQ getSeed
ECGroup	equals getBaseDigest getBasePoint getBaseSeed getCofactor getCurve getDigest getECPointFactory getEncodedPointLength getField getFieldElementFactory getOrder getSeed getVersion hashCode isVerifiableRandomBasePoint
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

Table 1 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Parameter Generation:	
AlgParamGenerator	generate getCryptoModule initGen initVerify verify
Key Generation:	
KeyGenerator	clearSensitiveData generate getCryptoModule initialize
KeyPairGenerator	clearSensitiveData clone generate getCryptoModule initialize
Keys:	
DHPrivateKey	clearSensitiveData clone getAlg getCryptoModule getParams getX
DHPublicKey	clearSensitiveData clone getAlg getCryptoModule getParams getY
DSAPrivateKey	clearSensitiveData clone getAlg getCryptoModule getPArams getX
DSAPublicKey	clearSensitiveData clone getAlg getCryptoModule getParams getY

Table 1 Services Available to the Crypto User and Crypto Officer Roles (continued)

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

Table 1 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Keys: (continued)	
PrivateKey	clearSensitiveData clone getAlg getCryptoModule
PublicKey	clearSensitiveData clone getAlg getCryptoModule isValid
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
Key Derivation:	
KDF	clearSensitiveData clone generate getCryptoModule

Table 1 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
Random Number Generation:	
SecureRandom	autoseed **clearSensitiveData** getCryptoModule newInstance nextBytes selfTest setAlgorithmParams setOperationalParameters setSeed
Other Services:	
AlgListener	asymCipher domainParams kdf keyAgree keyGenerator keyPairGenerator keyWrapCipher mac messageDigest paramGenerator privateKey publicKey secretKey secureRandom signature symCipher
BigNum	getBitLength toOctetString
CryptoModule	getDeviceType getKeyBuilder getModuleOperations newAlgInputParams newAlgParamGenerator newAsymmetricCipher newKDF newKeyAgreement newKeyGenerator newKeyPairGenerator newKeyWrapCipher newMAC newMessageDigest newSecureRandom newSignature newSymmetricCipher
JCMCloneable	clone

Table 1 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 getVersionString initFIPS140RolePINs isFIPS140Compliant newCryptoModule resetFIPS140RolePIN setEntropySource setFIPS140RolePIN
ModuleOperations	perform
PasswordKey	clearSensitiveData clone getAlg getCryptoModule getKeyData getPassword
SelfTestEvent	getTestId getTestName
SelfTestEventListener	failed finished forcedToFail 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 186-3 RSA key generation method.

The module employs a FIPS-approved HMAC Deterministic Random Bit Generator (HMAC DRBG SP 800-90A) 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 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 2 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	HMAC DRBG entropy, strength, and 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

1.5.5 Key Storage

JCM does not provide long-term cryptographic key storage. Storage of keys is the responsibility of the user of JCM.

The following table shows how the storage of keys and CSPs are handled. The Crypto User and Crypto Officer roles have equal and complete access to all keys and CSPs.

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 SHA1 and SHA2 keys	In volatile memory only (plaintext)
EC public keys	In volatile memory only (plaintext)
EC private keys	In volatile memory only (plaintext)
DH public key	In volatile memory only (plaintext)
DH private key	In volatile memory only (plaintext)
RSA public key	In volatile memory only (plaintext)
RSA private key	In volatile memory only (plaintext)
DSA public key	In volatile memory only (plaintext)
DSA private key	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)

1.6 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 4 JCM FIPS 140-2 approved Algorithms

Algorithm Type	Algorithm	Validation Certificate	
		6.1	6.1.1.0.1
Asymmetric Cipher	RSA	Non-Approved (Allowed in FIPS mode for key transport*)	
Key Agreement Primitives	Diffie-Hellman (primitives only) EC Diffie-Hellman (primitives only)	Non-Approved (Allowed in FIPS mode)	
Key Derivation	Password-based	Vendor Affirmed (Approved in FIPS mode for key storage**)	
	KDFTLS10***	39	39
	KDFTLS12**** with SHA-256, SHA-384, SHA-512	39	39
Message Authentication Code	HMAC*****	1378	1378
Message Digest	SHA-1, SHA-224, SHA-256, SHA- 384, SHA-512	1938	1938
	SHA-512/224, SHA-512/256	1938	1938
Random Bit Generator	HMAC DRBG	273	273
Signature*****	RSA X9.31, PKCS #1 V.1.5, RSASSA-PSS	1154	1154
	DSA	701	701
	ECDSA	357	357
Symmetric Cipher	AES (ECB, CBC, CFB, OFB, CTR, CCM, GCM, XTS) [128, 192, 256 bit key sizes]	2249	2249
	Triple-DES****** (ECB, CBC, CFB, OFB)	1408	1408

*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 "Crypto User Guidance" on page 25.

***Key Derivation in TLS for versions 1.0 and 1.1.

****Key Derivation in TLS for versions 1.2.

*****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:" on page 27.

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.

- BPS
- DES
- DESX
- ECIES
- Non-approved RNG (FIPS 186-2)
- Dual EC DRBG
- HMAC-MD5
- MD2
- MD4
- MD5[1]
- RC2® block cipher
- RC4® stream cipher
- RC5® block cipher
- RSA Keypair Generation MultiPrime (2 or 3 primes)
- RIPEMD160
- Shamir Secret Sharing.

[1]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
- HMAC DRBG Self-Test
- EC DRBG Self-Test
- HMAC-SHA1 software integrity check
- RSA Signature Generation KAT
- RSA Signature Verification KAT
- DSA Sign/Verify Test
- ECDSA Sign/Verify Test
- Non-approved RNG KAT (FIPS 186-2).

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 ECDSA key pair.
- Continuous Random Number Generator (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 random number generators (HMAC DRBG, Non-approved RNG (FIPS 186-2), EC DRBG, non-approved entropy source).

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 compliance with FIPS 140-2 Level 1 requirements, the module must be loaded using the following methods with the specified arguments:

For the Android platform:

```
com.rsa.crypto.jcm.ModuleLoader.load(File jarFile, int
securityLevel, int katStrategy, SelfTestEventListener
selfTestListener)
```

- where `jarFile` is the module JAR file, `securityLevel` is the value 1 (specified as the constant `ModuleConfig.LEVEL_1`) and `katStrategy` is the value 0 (specified as the constant `ModuleConfig.ON_LOAD`).

For all other platforms:

```
com.rsa.crypto.jcm.ModuleLoader.load(int securityLevel, int
katStrategy, SelfTestEventListener selfTestListener)
```

- where `securityLevel` is the value 1 (specified as the constant `ModuleConfig.LEVEL_1`) and `katStrategy` is the value 0 (specified as the constant `ModuleConfig.ON_LOAD`).

Using the specified `securityLevel` ensures that the module is loaded for use in a FIPS 140-2 Level 1 compliant way. Using the specified `katStrategy` value ensures that all module self-tests are run during module start-up, as required by FIPS 140-2.

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

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

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 4, “JCM FIPS 140-2 approved Algorithms,” on page 21.
- 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 feedback 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. It must be generated internally.
- In case the module’s power is lost and then restored, the key used for the AES GCM encryption/decryption shall be re-distributed.
- RSA keys used for signing shall not be used for any other purpose other than digital signatures.
- 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).
- The bit length of the input parameter to the Diffie-Hellman primitives method must be between 1024 and 2048 bits. Diffie Hellman shared secret provides between 80 bits and 112 bits of encryption strength[2].

[2]Using the minimum allowed modulus size, the minimum strength of encryption provided is 80 bits.

- Bit lengths for an HMAC key must be one half of the block size.
- For RSA digital signature generation the HMAC DRBG must be used.
- EC key pairs must have domain parameters from the set of NIST-recommended named curves (P-192, P-224, P-256, P-384, P-521, B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, and K-571). 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 80 bits and 256 bits of encryption strength.
- When using an approved random number generator to generate keys or DSA parameters, the random number generator's requested security strength must be at least as great as the security strength of the key being generated. That means that the HMAC DRBG with an appropriate strength must be used. For more information on requesting the random number generator security strength, see the relevant API *Javadoc*.
- When using an approved random number generator, 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.
- SHA1 is deprecated for the generation of digital signatures from 2011 to 2013, and will be disallowed after 2013.
- Only FIPS 140-2 approved random number generators may be used for generation of keys (asymmetric and symmetric).
- RSA keys shall have a modulus of size 1024, 2048 or 3072 bits, and shall have a public exponent of at least 65537.
- RSA key pairs shall be generated according to FIPS 186-3 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).
- DSA parameters shall be generated according to FIPS 186-3 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 following restrictions apply to the use of PBKDF:
  - The minimum password length is 10 characters, which has a strength of approximately 80 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.[3]
  - 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. Using the minimum allowed modulus size, the minimum strength of encryption provided is 80 bits.
- 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".

[3]For more information see nist-sp800-132.pdf

2.3.2 Crypto User Guidance on Obtaining Assurances for Digital Signature Applications

The module has added support for the FIPS 186-3 standard for digital signatures. The following gives an overview of the assurances required by FIPS 186-3.

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 186-3 requirements for signatories and verifiers and the corresponding module capabilities and recommendations.

Table 5 Signatory Requirements

FIPS 186-3 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 186-3 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 186-3. For the JCM API, `com.rsa.crypto.AlgParamGenerator.verify()` For ECDSA, use the NIST recommended curves as defined in section 2.3.1.
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 random number generator 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 6 Verifier Requirements

FIPS 186-3 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 186-3. 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.

For more details on the requirements, see the *FIPS 186-3 and NIST Special Publication 800-89*.

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. JCM provides a DRBG implementation that should be used for key generation, the HMAC DRBG.

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's security strength.

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

Table 7 Generated Key Sizes and Strength

Key Type	Key Sizes	Security Strength	Required DRBG Security Strength (Entropy)
AES Key	128, 192, 256	128, 192, 256	128, 192, 256
Triple-DES 3-Key	168	112	112
RSA Key Pair	1024, 2048, 3072, 4096	80, 112, 128, 128	80, 112, 128, 128
DSA Key Pair	1024, 2048, 3072, 4096	80, 112, 128, 128	80, 112, 128, 128
EC Key Pair	160, 224, 256, 384, 521	80, 112, 128, 192, 256	80, 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 1.5.5 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" on page 24 **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 JCM and their definitions.

Table 8 Acronyms used with JCM

Acronym	Definition
3DES	Refer to Triple-DES
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 cryptosystem. 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.
DES	Data Encryption Standard. A symmetric encryption algorithm with a 56-bit key.
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.

Table 8 Acronyms used with JCM (continued)

Acronym	Definition
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.
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.
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.
MD4	A message digest algorithm which implements a cryptographic hash function, created by Rivest.
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.

Table 8 Acronyms used with JCM (continued)

Acronym	Definition
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.
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.
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.
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.